TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Vizualizace automatů a regulárních výrazů (Online podpora výuky)

Finite automata and regular expressions visualisation (Online support for students)

Diplomová práce

Autor: **Bc. Petra Černíková**

Vedoucí práce: Ing. Lenka Kosková Třísková

V Liberci 8. 5. 2012

Zadání práce

Prohlášení

Byla jsem seznámena s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití jsem si vědoma povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Chtěla bych na tomto místě poděkovat především vedoucí mé diplomové práce Ing. Lence Koskové Třískové za ochotu, trpělivost a cenné rady při psaní této práce. Také bych chtěla poděkovat své rodině a přátelům za podporu nejen při tvorbě této práce, ale i během celého mého studia.

Abstrakt

Cílem této diplomové práce je vytvořit webovou aplikaci, která bude sloužit pro podporu výuky. Aplikace musí umožnit uživateli zadat konečný automat, a to buď ručně, nebo ze souboru. Zadaný automat musím přehledně vykreslit. Dále musím automat minimalizovat a celý proces zobrazit krok za krokem. Nakonec musím konečný automat převést na regulární výraz a opět postup zobrazit, navíc s možností krokovat. Zobrazení algoritmů musí být dostatečné názorné, aby studenti, kteří budou aplikaci používat, lépe pochopili probíranou látku.

Práce je rozdělena do dvou částí. V první se zabývám nezbytnou teorií konečných automatů, jejich reprezentací, minimalizací a převodů na regulární výraz. V druhé praktické části představuji své řešení dané problematiky.

Klíčová slova:

Abeceda, slovo, konečný automat, minimalizace, převod na regulární výraz.

Abstract

The aim of this diploma thesis is to create web application, which will serve to support teaching. Application must allow the user to specify a finite state machine, either manually or from a file. The specified machine must be clearly draw. The machine must be minimized and the process must be shown step by step. Finally the finite state machine must be converted to a regular expression and the procedure must again be displayed, with the additional option to step through. Views of algorithms must be sufficiently illustrative, so that the students, who will use this application, would better understand the discussed lesson.

The work is divided into two parts. In the first part I deal with the necessary theory of finite automata, their representations, minimization and transfers to a regular expression. In the second part I present my solution to the issue.

Keywords

Alphabeth, word, finite automata, minimization, transfer to a regular expression.

Obsah

Seznam symbolů, zkratek a termínů

Σ – konečná neprázdná množina vstupních symbolů (abeceda)

ε – prázdné slovo

Q – konečná neprázdná množina stavů

q – stav konečného automatu

δ – přechodová funkce

q_0 – počáteční stav

F – konečná neprázdná množina rozpoznávacích stavů

I – množina počátečních stavů

δ^* – zobecněná přechodová funkce

$P(Q)$ – množina všech možných podmnožin vytvořených z prvků množiny Q

L – formální jazyk nad abecedou Σ

$L(A)$ – jazyk rozpoznávaný automatem A

K – prvek z množiny $P(Q)$

$E(q)$ – množina stavů, do kterých může automat přejít ze stavu q pomocí ε přechodů

$E(K)$ – všechny množiny $E(q)$

$RJ(\Sigma)$ – třída regulárních jazyků nad abecedou Σ

$RV(\Sigma)$ – množina regulárních výrazů nad abecedou Σ

PHP – rekurzivní zkratka PHP: Hypertext Preprocessor

XML – eXtensible Markup Language

SCXML – State Chart eXtensible Markup Language

SVG – Scalable Vector Graphics

Úvod

Mým úkolem v této práci je vytvořit webovou aplikaci, která bude sloužit studentům předmětu Automaty a formální jazyky. Hlavním úkolem je přehledně vykreslit zadaný konečný automat, provést jeho minimalizaci a převod na regulární výraz. Tyto algoritmy musím uživateli ukázat krok za krokem, buď jako animaci nebo s možností krokovat.

Důvodů, proč jsem si toho téma vybrala, je hned několik. Téma konečných automatů mne zaujalo už během bakalářského studia, protože se s nimi setkáváme v běžném životě víc, než se na první pohled zdá. Díky této práci mohu do problematiky blíže proniknout. Dalším důvodem je použitelnost aplikace, neboť bude sloužit mým nynějším kolegům i těm, kteří přijdou po nich, k pochopení procesu minimalizace a převodu na regulární výraz.

V první polovině práce se zabývám teoretickým základem práce. Vysvětlím pojmy jako abeceda, slovo, konečný automat. Popíši rozdíl mezi deterministickým a nedeterministickým konečným automatem. Budu se zabývat možnostmi reprezentace těchto automatů. Dále se budu věnovat procesu minimalizace a nakonec vysvětlím algoritmus převodu na regulární výraz.

V druhé polovině představím vlastní řešení a technologie, které jsem při tvorbě využila. Seznámím čtenáře s rozhraním a možnostmi aplikace, kterou jsem navrhla. Ukáži na příkladech, jak vypadá ztvárnění jednotlivých algoritmů.

1 Teoretická část

1.1 Jazyk a jeho konečná reprezentace

Začneme definicí základních pojmů abeceda, slovo a jazyk.

1.1.1 Abeceda

Pod pojmem abeceda si jistě každý představí písmena a až ž, tak jak jsme se je učili v prvním ročníku základní školy. V teorii formálních jazyků definujeme abecedu obecněji, jako libovolnou konečnou neprázdnou množinu prvků a značíme Σ. Prvky nazýváme symboly a mohou to být písmena, znaky apod. Příkladem abecedy je množina $\Sigma = \{0,1\}$, která má dva symboly 0 a 1.

1.1.2 Slovo

Všechny konečné posloupnosti prvků, které můžeme vytvořit z abecedy Σ, nazýváme slova nad abecedou Σ nebo také řetězce nad abecedou Σ. Množinu všech slov nad abecedou Σ značíme Σ^*. V množině Σ^* připouštíme i prázdné slovo „“, které značíme ε. Množinu, kde nejsou prázdná slova značíme Σ^+. Platí $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. Jak množina Σ^*, tak množina Σ^+, jsou spočetné. To znamená, že jednotlivá slova těchto množin mohu očíslovat přirozenými čísly.

Délka slova označuje počet prvků, ze kterých se slovo skládá. Například, pokud máme slovo $u = aba$, potom délka slova u (značíme $|u|$) je rovna 3. Prázdné slovo ε má délku 0. Pokud nás zajímá počet výskytů určitého prvku, potom teto počet značíme $|u|^a = 2$, kde a je prvek, který nás zajímá.

1.1.3 Operace nad slovy

Důležitou operací nad slovy je jejich zřetězení. Máme-li dvě slova $u = 001$ a slovo $v = 010$, potom zřetězením těchto slov získáme $u.v = 001010$. Je tedy zřejmé, že pro délku zřetězení platí $|u.v| = |u| + |v|$. Pro zřetězení libovolného slova u a prázdného slova ε platí $u\varepsilon = \varepsilon u = u$. Výraz u^n označuje n-násobné zřetězení slova u. Například pro slovo $u = 001$ platí $u^2 = u.u = 001001$ a $u^0 = \varepsilon$. Operace zřetězení je asociativní, pro zřetězení slov u, v a w platí $u.v.w = (u.v).w = u.(v.w)$.

Jako reverzní slovo u^R označujeme slovo, kde jsou jeho symboly zapsány v opačném pořadí než u slova u. Jestliže $u = 001$, potom $u^R = 100$.

Dále definujeme pojmy podslovo, prefix a sufix. Slovo *v* je podslovem slova *u*, právě tehdy když je možné najít taková slova *x* a *y*, pro které platí $u = x.v.y$. Potom *x* je prefixem a *y* je sufixem slova *u*. V případě, že $x \neq \varepsilon$ případně $y \neq \varepsilon$, nejedná se tedy o prázdná slova, potom říkáme, že *x* je vlastní prefix případně *y* je vlastní sufix.

1.1.4 Jazyk

Pokud Σ je abeceda, potom libovolnou podmnožinu Σ^* nazýváme formální jazyk (dále jen jazyk) *L* nad abecedou Σ, tedy $L \subseteq \Sigma^*$. Jazyk nad abecedou Σ může obsahovat buď všechna možná slova z Σ^*, nebo žádné či jen některé. Jako příklad použiji abecedu $\Sigma = \{0,1\}$, jazyk nad touto abecedou může být například $L = \{\varepsilon,0,1,00,11,10\}$.

1.1.5 Operace nad jazyky

Vzhledem k tomu, že jazyk je množina, můžeme pro něj definovat operace jako sjednocení, průnik, rozdíl a doplněk (komplement) jazyka. Další operace jako zřetězení, i-tá mocnina, iterace a zrcadlový obraz jazyka můžeme použít díky faktu, že prvky jazyka jsou řetězce.

Z prvního typu operací stojí za zmínku doplněk. Jazyk L_1 je doplněk jazyka L_2, pokud platí $L_1 = \Sigma^* - L_2$.

Zřetězením jazyka L_1 a L_2 vznikne jazyk $L_1.L_2 = \{u.v | u \in L_1, v \in L_2\}$. Dále platí pro zřetězení s prázdnou množinou $\varnothing.L = L.\varnothing = L$ a pro zřetězení s jazykem obsahujícím jen prázdné slovo $\{\varepsilon\}.L = L.\{\varepsilon\} = L$. Operace zřetězení je asociativní, ale již není komutativní, platí $L_1.L_2.L_3 = (L_1.L_2).L_3 = L_1.(L_2.L_3)$ a $L_1.L_2 \neq L_2.L_1$.

U i-té mocniny jazyka L platí $L^0 = \{\varepsilon\}$, $L^{i+1} = L.L^i$ pro $i \in N_0$.

Iteraci jazyka *L* definujeme jako $L^* = \bigcup_{i=0}^{\infty} L^i$ a pozitivní iteraci jako $L^+ = \bigcup_{i=1}^{\infty} L^i$. Pro iterace také platí $L^* = \{\varepsilon\}.L^+$ a $L^+ = L.L^* = L^*.L$.

Zrcadlový obraz jazyka vychází z definice zrcadlového obrazu slova a platí tedy $L^R = \{u | \exists v \in L : u = v^R\}$.

1.2 Deterministický konečný automat

Konečný automat, jak již z názvu vyplývá, je automat nabývající konečný počet stavů. Automat reaguje na vstup tvořený prvkem z abecedy. U každého stavu je jednoznačně definováno, do kterého stavu přejde automat při příchodu určitého symbolu. Pokud budeme uvažovat nad souvislostí konečného automatu se slovy a jazyky, potom automat slouží k rozhodnutí, zda konkrétní zadané slovo patří do jazyka nebo ne. Automat má definovaný počáteční stav, tedy stav kde začít a také stav rozpoznávací (někdy uváděný jako koncový nebo přijímací). Převede-li slovo automat z počátečního stavu do některého rozpoznávacího, potom je slovo prvkem jazyka. Se stavovým automatem se můžeme setkat v reálném životě na každém kroku, můžeme jej využít k modelování chování například automatu na nápoje, mechanizmu automatického otevírání dveří, světelné křižovatky apod. Příkladů bychom našli mnoho.

Konečný deterministický automat se definuje jako uspořádaná pětice $A = (Q,\Sigma,\delta,q_0,F)$. Q je konečná neprázdná množina stavů, Σ je konečná neprázdná množina vstupních symbolů (neboli abeceda, která byla definována v kapitole 1.1.1), δ realizuje zobrazení $Q \times \Sigma \to Q$, toto zobrazení nazýváme přechodovou funkcí, q_0 je počáteční stav a platí $q_0 \in Q$ a nakonec F je množina rozpoznávacích stavů a platí $F \subseteq Q$. Konečný automat můžeme reprezentovat grafem (stavovým diagramem), přechodovou tabulkou nebo stavovým stromem.

1.2.1 Reprezentace konečného deterministického automatu

Graf, který pro reprezentaci využíváme, je orientovaný a ohodnocený graf. Vrcholy grafu jsou stavy automatu, hrany jsou orientovány podle přechodových funkcí a jsou ohodnoceny vstupními symboly. Vstupní stav je nejčastěji označen šipkou, která do něj vstupuje. Rozpoznávací stavy jsou označeny dvojitým kruhem, také je možné značení šipkou, která ze stavu vystupuje.

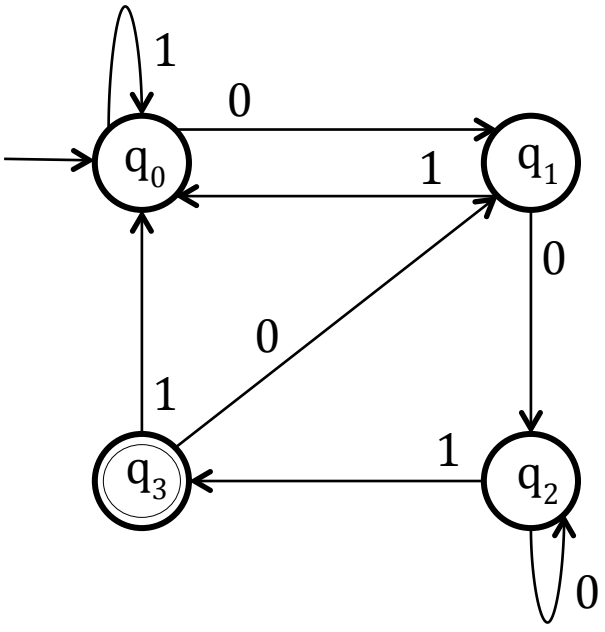
Přechodová tabulka zobrazuje přechodovou funkci δ automatu. V prvním řádku najdeme vstupní symboly a v prvním sloupci stavy automatu. Počáteční stav je označen šipkou směřující k tomuto stavu $\rightarrow$. Rozpoznávací stavy jsou označeny šipkou směřující ven z příslušného řádku $\leftarrow$. V případě, že stav je zároveň počáteční i rozpoznávací, pak je nejčastěji označen $\leftrightarrow$.

Než popíši reprezentaci pomocí stavového stromu, musím udělat malou odbočku a vysvětlit několik pojmů z teorie grafů, které budu využívat. Těmito pojmy jsou orientovaný strom, kořen, list. Orientovaný strom je souvislý acyklicky orientovaný

graf, jehož každé dva vrcholy jsou spojené právě jednou hranou. Mezi vrcholy grafu existuje jen jedna cesta. Kořen je vrchol grafu, do kterého nevedou žádné hrany (tedy nemá předky), hrany vedou pouze z něj. Takový vrchol se v grafu vyskytuje jen jednou. Oproti tomu list je vrchol grafu, ze kterého žádné hrany nevystupují (tedy nemá následovníky). Do jednoho listu vstupuje vždy jen jedna hrana.

Reprezentace konečného automatu pomocí stavového stromu znamená, že z každého vrcholu grafu kromě listů vystupuje vždy tolik hran, kolik je vstupních symbolů. Počáteční stav je kořen stromu a označuje se šipkou, která ze stavu vystupuje. Hrany mezi stavy a jejich následovníky jsou označeny podle přechodové funkce. Dostaneme-li se do stavu, který jsme již zpracovali, pak z toho stavu vytvoříme list. Reprezentace pomocí stavového stromu je možná jen u automatů, které mají všechny stavy dosažitelné. Dosažitelné stavy jsou takové, na které je možné dostat se procházením automatu z počátečního stavu pomocí přechodových funkcí z počátečního stavu. Rozpoznávací stavy označíme šipkou směřující od vrcholu.

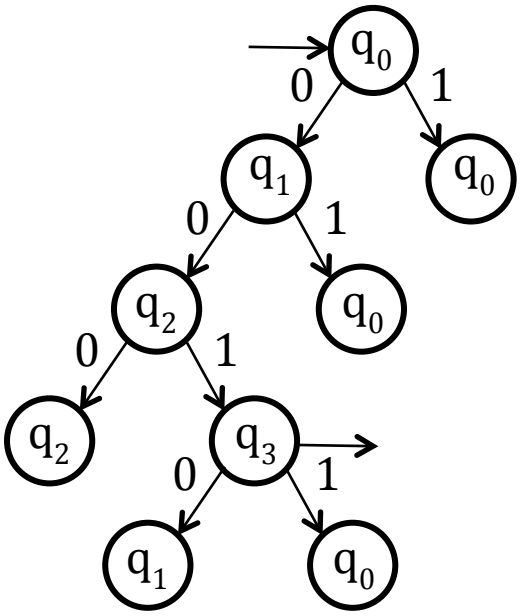
Příkladem konečného deterministického automatu je automat, který rozpoznává slova končící 001. Máme automat $A = (Q, \Sigma, \delta, q_0, F)$. $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $q_0 = q_0$, $F = \{q_3\}$, $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_0$, $\delta(q_1,0) = q_2$, $\delta(q_1,1) = q_0$, $\delta(q_2,0) = q_2$, $\delta(q_2,1) = q_3$, $\delta(q_3,0) = q_1$, $\delta(q_3,1) = q_0$.



Obrázek 1: Reprezentace pomocí grafu

Tabulka 1: Reprezentace automatu *A* pomocí přechodové tabulky

Stavy/Vstupní symboly	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
$\leftarrow q_3$	q_1	q_0



Obrázek 2: Reprezentace pomocí stavového stromu

1.2.2 Zobecněná přechodová funkce

Máme-li konečný automat $A = (Q, \Sigma, \delta, q_0, F)$ a přechodovou funkci $\delta : Q \times \Sigma \rightarrow Q$, potom zobecněnou přechodovou funkcí nazveme $\delta^* : Q \times \Sigma^* \rightarrow Q$ a definujeme ji jako:

1. $\delta^*(q, \varepsilon) = q, \ \forall q \in Q$ (1)
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a), \ \forall q \in Q, \ w \in \Sigma^*, \ a \in \Sigma$ (2)

Znamená to tedy, že automat *A* přejde z jednoho stavu pomocí slova *w* (to znamená postupným čtením slova *w* znak po znaku zleva doprava) do stavu jiného.

1.2.3 Jazyk rozpoznávaný konečným automatem

Slovo $w \in \Sigma^*$ je rozpoznané konečným automatem A, právě tedy když $\delta^*(q_0, w) \in F$. Jinými slovy je slovo w rozpoznatelné, pokud automat přejde z počátečního stavu postupně přes všechny symboly slova do některého z rozpoznávacích stavů.

Jazyk rozpoznávaný konečným automatem A nazýváme jazyk $L(A) = \{w \mid w \in \Sigma^* \wedge \delta^*(q_0, w) \in F\}$.

Existuje-li konečný automat A, takový že jazyk $L = L(A)$, potom říkáme, že jazyk L (nad abecedou Σ) je rozpoznatelný konečným automatem. Takový jazyk L nazýváme jazykem regulárním.

1.2.4 Nerodova věta

Nerodova věta se využívá k rozhodnutí, zda zadaný jazyk je, nebo není rozpoznatelný konečným automatem.

Je-li Σ konečná abeceda a $\sim$ je relace ekvivalence na Σ^*. Relaci $\sim$ nazýváme pravou kongruencí, pokud pro všechna slova $u, v, w \in \Sigma^*$ platí: Jestliže $u \sim v$, potom $uw \sim vw$. Relace ekvivalence $\sim$ na množině M je konečného indexu, jestliže rozklad $M / \sim$ má konečný počet tříd.

Sama Nerodova věta potom zní: nechť L je jazyk nad konečnou abecedou Σ. Potom L je rozpoznatelný konečným automatem právě tehdy, když existuje pravá kongruence konečného indexu taková, že L je sjednocením jistých tříd rozkladu $\Sigma^* / \sim$.

1.3 Nedeterministický konečný automat

Nedeterministický konečný automat je uspořádaná pětice $A = (Q, \Sigma, \delta, I, F)$. Q je neprázdná konečná množina stavů. Σ je neprázdná konečná množina vstupních symbolů (abeceda). δ je přechodová funkce, která realizuje zobrazení $Q \times \Sigma \to P(Q)$. $P(Q)$ reprezentuje množinu všech možných podmnožin množiny Q. I je množina počátečních stavů a platí $I \subseteq Q$. F je množina rozpoznávacích stavů a platí pro ni $F \subseteq Q$.

Nedeterministické konečné automaty jsou vlastně zobecněné deterministické konečné automaty. Z tohoto důvodu je možné reprezentovat nedeterministický automat přechodovou tabulkou i grafem.

Na rozdíl od deterministického automatu není u nedeterministického jednoznačně určeno, do jakého stavu automat přejde z jednoho stavu přes jeden vstupní symbol. Automat může přejít z jednoho stavu přes jeden vstupní symbol do množiny stavů a tato množina může být i prázdná. Dalším rozdílem je, že nedeterministický automat může mít více než jeden vstupní stav.

1.3.1 Zobecněná přechodová funkce

Zobecněnou přechodovou funkci pro nedeterministický automat $A = (Q, \Sigma, \delta, I, F)$ popíšeme definicí $\delta^* : P(Q) \times \Sigma^* \to P(Q)$ a platí:

1. $\delta^*(K, \varepsilon) = K$, $\forall K \in P(Q)$, (3)
2. $\delta^*(K, wa) = \bigcup_{q \in \delta^*(K,w)} \delta(q, a)$, $\forall K \in P(Q)$, $w \in \Sigma^*$, $a \in \Sigma$. (4)

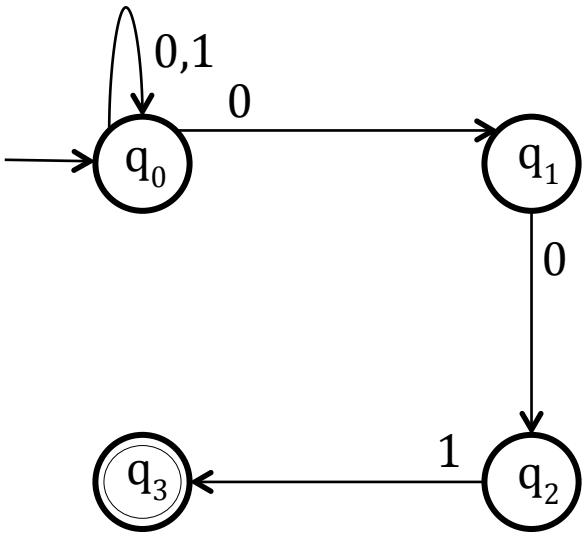
1.3.2 Jazyk rozpoznávaný nedeterministickým konečným automatem

Slovo $v = a_1 ... a_n \in \Sigma^+$ je přijímáno (rozpoznáváno) nedeterministickým automatem $A = (Q, \Sigma, \delta, I, F)$ právě tehdy, když existuje posloupnost stavů $q_1, q_2, ..., q_{n+1} \in Q$, kde $q_1 \in I$ a $q_{n+1} \in F$ a pro všechna $i = \{1, ..., n\}$ platí $q_{i+1} \in \delta(q_i, a_i)$. Jinými slovy, přejde-li automat přes symboly slova v z některého počátečního stavu do některého koncového. Prázdné slovo ε je přijímáno právě tehdy, když $I \cap F \neq \emptyset$, tedy některý počáteční stav, je zároveň i rozpoznávací.

Jazyk $L(A)$ rozpoznávaný nedeterministickým konečným automatem A je množina všech slov rozpoznávaných automatem, platí tedy $L(A) = \{w \in \Sigma^* \mid \delta^*(I, w) \cap F \neq \emptyset\}$.

Existuje-li takový nedeterministický konečný automat A, že jazyk $L = L(A)$, potom říkáme, že jazyk L (nad abecedou Σ) je rozpoznatelný nedeterministickým konečným automatem.

Jako příklad uvedu automat, který rozpoznává slova končící 001. Tento automat tedy rozpoznává stejná slova jako předchozí deterministický konečný automat. Máme konečný automat $A = (Q, \Sigma, \delta, I, F)$, kde $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $I = \{q_0\}$, $F = \{q_3\}$, $\delta(q_0,0) = q_0$, $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_0$, $\delta(q_1,0) = q_2$, $\delta(q_2,1) = q_3$.



Obrázek 3: Reprezentace automatu pomocí grafu

Tabulka 2: Reprezentace automatu pomocí přechodové tabulky

Stavy/Vstupní symboly	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	-
q_2	-	q_3
$\leftarrow q_3$	-	-

1.4 Zobecněný nedeterministický konečný automat

Zobecněný nedeterministický automat definujeme jako uspořádanou pětici $A = (Q, \Sigma, \delta, I, F)$, kde Q je konečná množina stavů, Σ je konečná množina vstupních symbolů, I je konečná množina počátečních stavů, F je množina rozpoznávacích stavů a δ je přechodová funkce, která realizuje zobrazení: $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$.

Na rozdíl od nedeterministického automatu může zobecněný nedeterministický automat obsahovat ε přechody. Automat má možnost přejít z jednoho stavu do druhého, aniž by přečetl vstupní symbol.

Zobecněný nedeterministický automat je možné reprezentovat pomocí grafu.

1.4.1 Zobecněná přechodová funkce

Nejprve pro zobecněný nedeterministický automat $A = (Q, \Sigma, \delta, I, F)$ definujeme množinu, do které může automat přejít ze stavu q jen po ε přechodech a označíme ji $E(q)$ pro všechna $q \in Q$. A dále definujeme zobecnění množiny $E(q)$ množinu $E(K)$, pro množinu stavu $K \subseteq Q$. Množina $E(q)$ je nejmenší množina, pro kterou platí:

1. $q \in E(q)$, (5)
2. je-li $q_1 \in E(q)$ a $q_2 \in \delta(q_1, \varepsilon)$, potom $q_2 \in E(q)$. (6)

Pro množinu $E(K)$, kde $K \subseteq Q$, platí $E(K) = \cup_{q \in S} E(q)$.

Jinými slovy: E je funkce, která k danému stavu q přiřadí množinu stavů, na které je možné se dostat ze stavu q pomocí ε.

Nyní definujeme zobecněnou přechodovou funkci $\delta^* : P(Q) \times \Sigma^* \to P(Q)$, a to následovně:

1. $\delta^*(K, \varepsilon) = E(K)$, kde $K \subseteq Q$, (7)
2. $\delta^*(K, wa) = E(\cup_{q \in \delta^*(S,w)} \delta(q, a))$, kde $K \subseteq Q$, $w \in \Sigma^*$ a $a \in \Sigma$. (8)

1.4.2 Jazyk rozpoznávaný zobecněným nedeterministickým automatem

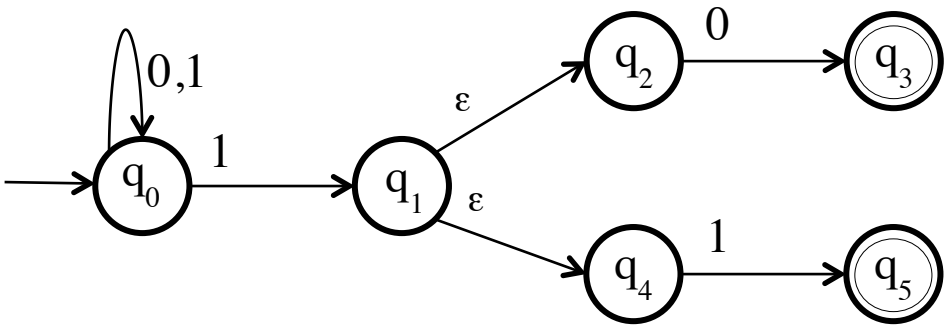
Jazyk rozpoznávaný zobecněným nedeterministickým automatem definujeme stejně jako u nedeterministického automatu. Platí: $L(A) = \{w \in \Sigma^* \mid \delta^*(I, w) \cap F \neq \varnothing\}$. Pravidla pro rozpoznání slova jsou obdobná jako u nedeterministického automatu. Slovo je rozpoznáno, přejde-li automat pomocí symbolů tohoto slova z některého počátečního do některého rozpoznávacího stavu.

1.4.3 Převod zobecněného nedeterministického automatu na nedeterministický

Každý zobecněný nedeterministický automat můžeme převést na nedeterministický automat (bez ε přechodů).

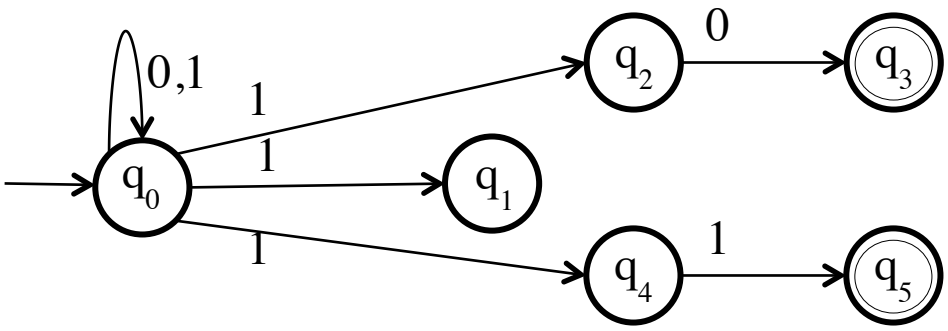
1. Spočteme funkci *E* pro všechny stavy.
2. Pro každý stav q, q_1 a symbol $a \in \Sigma$, kde platí $\delta(q_1, a) = q$, vytvoříme novou přechodovou funkci nedeterministického automatu tak, že do ní zahrneme $\delta(q_1, a) = q$ a přidáme $\delta(q_1, a) = q_2$ pro každý stav $q_2 \in E(q)$. Jinými slovy, existuje-li přechod z jednoho stavu do druhého přes nějaký vstupní symbol a z druhého stavu lze přejít ε přechodem do třetího stavu, potom musíme do přechodových funkcí přidat přechod z prvního stavu do třetího přes vstupní symbol, pomocí kterého přecházel automat z prvního stavu na druhý.
3. Počáteční stavy nedeterministického automatu budou všechny počáteční stavy zobecněného nedeterministického plus ty, na které se dostaneme z původních počátečních stavů pomocí ε přechodů.

Příkladem je automat $A = (Q, \Sigma, \delta, I, F)$, který rozpoznává slova končící 10 nebo slova končící 11. Automat zobrazíme pomocí grafu následovně:



Obrázek 4: Zobecněný nedeterministický automat reprezentovaný grafem

Podle předepsaného postupu převedeme zobecněný nedeterministický automat *A* na nedeterministický automat *B*, který zobrazíme opět pomocí grafu:



Obrázek 5: Nedeterministický automat reprezentovaný grafem

Jak je vidět na obrázku 5, při převodu na nedeterministický automat jsme odstranili ε přechody mezi stavy q_1 , q_2 a q_1 , q_4 a přidali jsme přechod z q_0 přes symbol 1 na stav q_2 a přechod z q_0 přes symbol 1 na stav q_4 .

1.5 Převod nedeterministického konečného automatu na deterministický

Každý jazyk, který je rozpoznatelný nedeterministickým konečným automatem A_1, je rozpoznatelný odpovídajícím deterministickým automatem A_2 (platí $L(A_1) = L(A_2)$). Jinými slovy, ke každému nedeterministickému automatu existuje ekvivalentní deterministický.

Jako důkaz předchozího tvrzení lze použít podmnožinovou konstrukci, díky které můžeme z nedeterministického automatu $A_1 = (Q_1, \Sigma, \delta_1, I, F_1)$ vytvořit deterministický automat $A_2 = (Q_2, \Sigma, \delta_2, q_0, F_2)$. Vezme nedeterministický automat a vytvoříme podmnožiny stavů, do kterých automat přejde přes jednotlivé symboly. Ze získaných podmnožin stavů vytvoříme stavy nového deterministického automatu a v tomto automatu bude pro každou dvojici (stav, vstupní symbol) právě jeden přechod. Získané podmnožiny můžeme zapisovat, buď do stromu nebo tabulky.

1.5.1 Stromový algoritmus

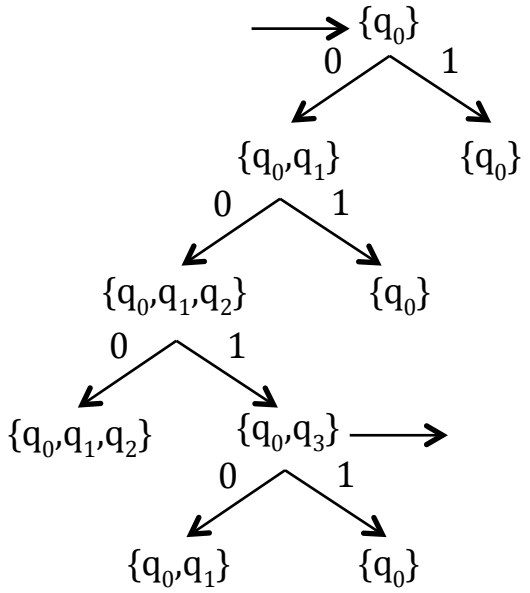
1. V prvním kroku vytvoříme kořen stromu. Ten obsahuje množinu počátečních stavů z množiny I automatu A_1.
2. Z kořene stromu vytvoříme větve a ohodnotíme je vstupními symboly. Větví bude právě tolik, kolik máme vstupních symbolů. Do obsahu nově vzniklých uzlů vložíme podmnožiny stavů. Ty budou obsahovat stavy, do kterých se dostaneme přes konkrétní vstupní symbol ze všech stavů množiny I (z kořene stromu).
3. Pokračujeme obdobně jako v kroku 3. Z nově vzniklých podmnožin vytvoříme větve podle vstupních symbolů. Obsah vrcholu získáme jako podmnožinu stavů, do kterých automat přejde z jednotlivých stavů předchozí podmnožiny pomocí vstupních symbolů. Pokud se nově vzniklá podmnožina rovná některé již vytvořených podmnožin, potom u ní již nevytváříme větve. Nezáleží přitom na pořadí stavů v podmnožině, protože je jednoznačně určena svými prvky nikoli jejich pořadím.
4. Krok 2 opakujeme, dokud vznikají nějaké nové podmnožiny stavů.
5. Po vytvoření celého stromu projdeme všechny podmnožiny a obsahují-li některý stav z množiny F_1, potom celou podmnožinu (vrchol) označíme šipkou z ní vystupující. Z takto označených podmnožin vzniknou rozpoznávací stavy deterministického automatu. Podmnožinu, která tvoří kořen stromu označíme šipkou do ní vstupující. V novém automatu bude sloužit jako počáteční stav.

6. Všechny vzniklé podmnožiny dále přepíšeme symboly $q_0, q_1, ..., q_{n-1}$. Tyto symboly tvoří množinu stavů nového deterministického automatu. Označení musí být jednoznačné, tedy každá unikátní podmnožina má také unikání symbol. Jak jsem již zmínila počáteční stav q_0 automatu A_2 vznikne z kořene stromu. Množina rozpoznávacích stavů F_2 vznikne z vrcholů označených vystupující šipkou.
7. Nakonec vytvoříme nové přechodové funkce tak, že vezme stav q_i a jeho potomka q_j, na kterého se dostaneme přes vstupní symbol s. Přechodové funkce budou ve tvaru $\delta_2(q_i, s) = q_j$. Opakujeme pro všechny dvojice stavů a vstupních symbolů.

Při použití přechodové tabulky je algoritmus převodu shodný, jen podmnožiny nezapisujeme do stromu, ale do tabulky.

Jako příklad použiji nedeterministický automat zmíněný v předchozí kapitole. Automat $A = (Q, \Sigma, \delta, I, F)$, kde $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $I = \{q_0\}$, $F = \{q_3\}$, $\delta(q_0,0) = q_0$, $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_0$, $\delta(q_1,0) = q_2$, $\delta(q_2,1) = q_3$.

Strom, který vznikne použitím stromového algoritmu vypadá následovně:



Obrázek 6: Stavový strom pro deterministický automat

Přechodová tabulka vzniklá použitím algoritmu:

Tabulka 3: Vytvořená tabulka deterministického automatu

Stavy/Vstupní symboly	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
$\leftarrow \{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0\}$

Výsledný deterministický konečný automat reprezentovaný pomocí tabulky:

Tabulka 4: Přechodová tabulka deterministického automatu

Stavy/Vstupní symboly	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
$\leftarrow q_3$	q_1	q_0

1.5.2 Převod zobecněného nedeterministického automatu na deterministický

Pokud máme zobecněný nedeterministický automat, je samozřejmě možné převést ho nejprve na nedeterministický a ten potom převést na deterministický. Existuje ovšem i algoritmus, který dokáže zobecněný nedeterministický automat převést rovnou na deterministický. Tento algoritmus je podobný jako při převodu nedeterministického automatu na deterministický. Při sestavování stromu, je nutné dát do kořene nejen množinu počátečních stavů I, ale do této množiny musíme přidat i stavy, do kterých se dostaneme ze stavů množiny I přes ε přechody. Dále při vytváření větví stromu, pokud do množiny stavů nově vzniklého vrcholu přidáme stav q, potom musíme přidat i množinu $E(q)$, tedy všechny stavy, na které se dostaneme ze stavu q přes ε přechody. Zbytek algoritmu již probíhá stejně jako u převodu nedeterministického automatu na deterministický.

1.6 Minimalizace konečného automatu

Při procesu minimalizace vycházíme z předpokladu, že dva automaty A_1 a A_2 jsou ekvivalentní právě tehdy, když rozpoznávají stejný jazyk. Platí $L(A_1) = L(A_2)$.

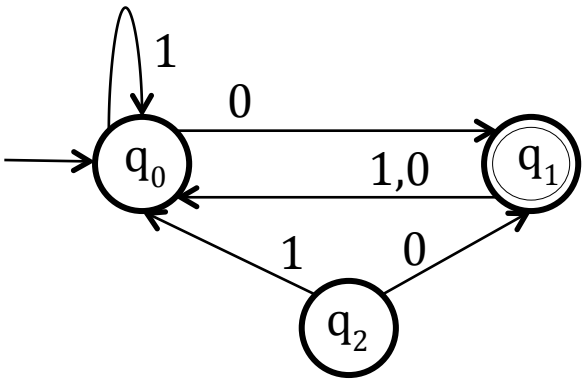
Konečný automat A je minimální právě tehdy, když k němu neexistuje ekvivalentní automat s menším počtem stavů. Pokud chceme konečný automat minimalizovat, musíme z něj odstranit nadbytečné stavy. Nadbytečnými stavy rozumíme stavy nedosažitelné a stavy ekvivalentní.

1.6.1 Nedosažitelné stavy a jejich odstranění

Nedosažitelné stavy jsou takové stavy, do kterých se při průchodu automatem z počátečního stavu nedostaneme. Tedy neexistuje žádná posloupnost vstupních symbolů (slov), která přesune automat z počátečního stavu do stavu nedosažitelného.

Naopak stav q automatu $A = (Q, \Sigma, \delta, q_0, F)$ je dosažitelný právě tehdy, když existuje takové slovo $w \in \Sigma^*$, že $\delta^*(q_0, w) = q$.

Nedosažitelný stav snadno poznáme, pokud máme konečný automat reprezentovaný grafem, protože v takovém grafu neexistují hrany směřující do nedosažitelných stavů.



Obrázek 7: Automat s nedosažitelným stavem

Na takto jednoduchém příkladu je jasně vidět, že stav q_2 je nedosažitelný, protože do něj nevedou žádné hrany. Pro složitější automaty bude lepší definovat algoritmus pro nalezení nedosažitelných stavů.

1.6.2 Algoritmus nalezení nedosažitelných stavů

1. Všechny stavy jsou neoznačené. Označíme počáteční stav jako dosažitelný.
2. Označíme jako dosažitelné všechny stavy, do kterých se můžeme dostat z počátečního stavu pomocí přechodových funkcí.

3. Procházíme všechny stavy označené jako dosažitelné, dostaneme-li se z těchto stavů pomocí přechodových funkcí na stav, který jsme zatím neoznačili, potom ho označíme jako dosažitelný. Opakujeme dokud nacházíme neoznačené stavy.
4. Stavy, které zbyly neoznačené jsou nedosažitelné.

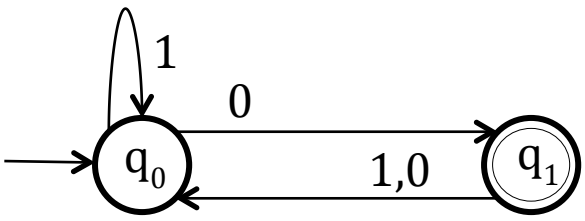
Nedosažitelné stavy můžeme odstranit, protože neovlivní rozpoznatelnost slova tedy ani rozpoznatelnost jazyka.

Máme-li automat $A = (Q, \Sigma, \delta, q_0, F)$ a množinu dosažitelných stavů P, pro kterou platí $P \subseteq Q$, potom existuje automat $B = (P, \Sigma, \delta_P, q_0, F \cap P)$, který je ekvivalentní s automatem A a neobsahuje nedosažitelné stavy. δ_P je parcializací přechodové funkce δ na množinu $P \times \Sigma$, jinými slovy pokud odstraňujeme nedosažitelný stav, potom odstraníme i přechodové funkce, které z nedosažitelného stavu vycházejí.

Je jasné, že všechny stavy, které automat projde při průchodu pomocí slova *w*, jsou dosažitelné. Platí tedy $\forall w \in \Sigma^* \quad \delta^*(q_0, w) = \delta_P^*(q_0, w)$. Pro množinu koncových stavů $F \cap P$ platí $\forall w \in \Sigma^* \quad \delta^*(q_0, w) \in F$ právě tehdy, když $\delta_P^*(q_0, w) \in F \cap P$. Je tedy zřejmé, že platí $L(A) = L(B)$.

Vezmeme předchozí příklad automatu s nedosažitelným stavem $A = (Q, \Sigma, \delta, q_0, F)$, kde $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $q_0 = q_0$, $F = \{q_1\}$, $\delta(q_0, 0) = q_1$, $\delta(q_0, 1) = q_0$, $\delta(q_1, 0) = q_0$, $\delta(q_1, 1) = q_0$, $\delta(q_2, 0) = q_1$, $\delta(q_2, 1) = q_0$.

Ekvivalentním automatem s dosažitelnými stavy je automat $B = (P, \Sigma, \delta_P, q_0, F \cap P)$, kde $P = \{q_0, q_1\}$, $\Sigma = \{0,1\}$, $q_0 = q_0$, $F \cap P = \{q_1\}$, $\delta_P(q_0, 0) = q_1$, $\delta_P(q_0, 1) = q_0$, $\delta_P(q_1, 0) = q_0$, $\delta_P(q_1, 1) = q_0$.



Obrázek 8: Automat *B* reprezentovaný grafem

1.6.3 Ekvivalentní stavy a jejich odstranění

Stav q_1 je ekvivalentní se stavem q_2 právě tehdy, když $\forall w \in \Sigma^*$ platí $\delta^*(q_1, w) \in F \Leftrightarrow \delta^*(q_2, w) \in F$. Pokud jsou stavy q_1 a q_2 ekvivalentní, potom můžeme tyto stavy sjednotit, aniž by se změnil jazyk rozpoznávaný automatem, tedy po sjednocení získáme automat ekvivalentní s původním automatem s menším počtem stavů.

Algoritmus, který je možné použít k nalezení a sjednocení ekvivalentních stavů je rozklad na množiny rozkladu.

1.6.4 Rozklad konečného automatu na množiny rozkladu

Na začátku máme definovaný deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů. Pro lepší přehlednost budeme výsledky rozkladu zapisovat do tabulky.

1. V prvním kroku rozdělíme množinu stavů na dvě podmnožiny. V první jsou všechny rozpoznávací stavy $R_{II} = F$, ve druhé ostatní stavy $R_I = F \setminus Q$.
2. Do prvního sloupce první tabulky nyní zapíšeme všechny stavy, přičemž stavy každé podmnožiny zapíšeme pod sebe. Pro lepší čitelnost je dobré obarvit řádky tabulky podle toho, do které podmnožiny stav patří. Do prvního řádku zapíšeme vstupní symboly z množiny Σ stejně, jako když vyplňujeme přechodovou tabulku.
3. Zbytek první tabulky doplníme podle přechodových funkcí, ale nebudeme zapisovat cílové stavy, nýbrž označení podmnožiny, ve které se cílový stav nachází.
4. V tomto kroku zkontrolujeme řádky, které spadají pod jednotlivé množiny. Pokud se řádky patřící do jedné podmnožiny nerovnají, pak musíme tuto množinu rozdělit na další podmnožiny.
5. Pokud v předchozím kroku došlo k rozdělení na další podmnožinu, potom musíme tabulku přepsat. Do prvního sloupce opět pod sebe sepíšeme stavy z podmnožin. Obarvíme řádky podle příslušných podmnožin a doplníme tabulku označením podmnožin podle toho, do které podmnožiny spadá cílový stav z přechodových funkcí.
6. Opakujeme kroky 4 a 5 tak dlouho, dokud je nutné dělit množinu stavů na další podmnožiny.

Výsledkem tohoto postupu jsou podmnožiny množiny Q, které nazýváme množiny (nebo také třídy) rozkladu, ve kterých jsou stavy vzájemně ekvivalentní. Tyto ekvivalentní stavy můžeme nyní sloučit. Pokud se v podmnožině vyskytuje počáteční stav, potom i nový stav vzniklý sloučením prvků této podmnožiny bude počáteční.

Celý postup lépe objasní následný příklad:

Máme automat $A = (Q, \Sigma, \delta, q_0, F)$, který neobsahuje nedosažitelné stavy, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0,1\}$, $q_0 = q_0$, $F = \{q_1\}$, $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_2$, $\delta(q_1,0) = q_1$, $\delta(q_1,1) = q_3$, $\delta(q_2,0) = q_1$, $\delta(q_2,1) = q_0$, $\delta(q_3,0) = q_0$, $\delta(q_3,1) = q_2$.

Reprezentace pomocí přechodové tabulky automatu *A* vypadá takto:

Tabulka 5: Reprezentace automatu *A* pomocí tabulky

Stavy/Vstupní symboly	0	1
$\rightarrow q_0$	q_1	q_2
$\leftarrow q_1$	q_1	q_3
q_2	q_1	q_0
q_3	q_0	q_2

Podle 1., 2. a 3. kroku vytvoříme následující tabulku:

Tabulka 6: Upravená přechodová tabulka podle kroku 1, 2 a 3

Množina	**Stavy/Vstupní symboly**	0	1
I	$\rightarrow q_0$	*II*	*I*
I	q_2	*II*	*I*
I	q_3	*I*	*I*
II	$\leftarrow q_1$	*II*	*I*

Vidíme, že řádek se stavem q_3 se nerovná ostatním v množině *I*, proto podle 4. a 5. kroku přidáme další množinu a přepíšeme tabulku:

Tabulka 7: Upravená přechodová tabulka po aplikaci kroku 4 a 5

Množina	**Stavy/Vstupní symboly**	0	1
I	$\rightarrow q_0$	*III*	*I*
I	q_2	*III*	*I*
II	q_3	*I*	*I*
III	$\leftarrow q_1$	*III*	*II*

Nyní se již řádky v jednotlivých podmnožinách rovnají a proto nebudeme v rozdělování pokračovat. V množině *I* se nachází dva stavy, které jsou ekvivalentní, jež sloučíme. Vzhledem k tomu, že stav q_0 je počáteční, bude proto i stav vzniklý sloučením počáteční. Stavy minimalizovaného automatu nazveme označením množin.

Tabulka 8: Přechodová tabulka minimalizovaného automatu

Stavy/Vstupní symboly	0	1
$\rightarrow I$	*III*	*I*
II	*I*	*I*
$\leftarrow III$	*III*	*II*

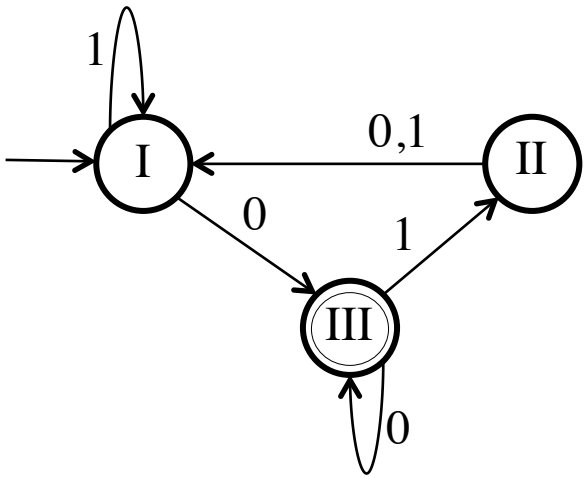
Výsledný minimalizovaný konečný automat, vypadá následovně: $A = (Q, \Sigma, \delta, q_0, F)$, kde $Q = \{I, II, III\}$, $\Sigma = \{0,1\}$, $q_0 = I$, $F = \{III\}$, $\delta(I,0) = III$, $\delta(I,1) = I$, $\delta(II,0) = I$, $\delta(II,1) = I$, $\delta(III,0) = III$, $\delta(III,1) = II$.

Minimalizace konečného automatu je velmi dobrým nástrojem pro zjištění, zda jsou dva automaty ekvivalentní. Platí, že pokud jsou dva automaty ekvivalentní, potom výsledkem jejich minimalizace jsou automaty lišící se nejvýše v názvech stavů těchto automatů. Abychom mohli sjednotit i názvy automatů a získali tak stejné automaty, zavedeme pojem normovaný tvar automatu.

1.7 Normovaný tvar

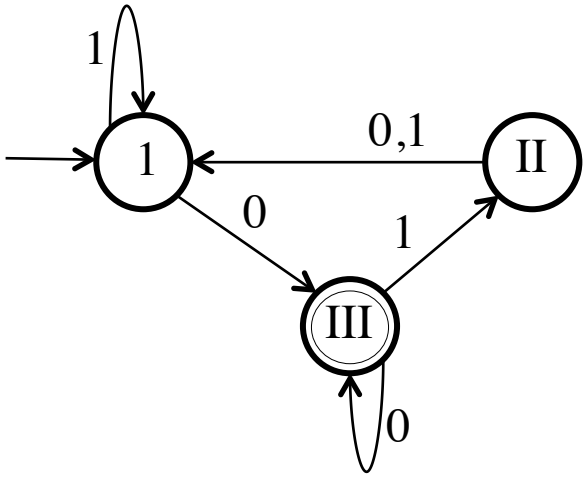
Při vytváření normovaného tvaru konečného automat přepisujeme názvy stavů automatu přirozenými čísly od 1 v pořadí, v jakém se na ně dostáváme z počátečního stavu pomocí přechodových funkcí. Jinými slovy, počáteční stav označíme 1, 2 potom označíme stav, na který se dostaneme přes první vstupní symbol z abecedy Σ a tak pokračujeme pro všechny vstupní symboly. Následně vezmeme stav označený jako 2 a označíme dalšími čísly stavy, na které se dostaneme přes vstupní symboly v pořadí, v jakém jsou v abecedě Σ. Postup opakujeme, dokud nejsou označeny všechny stavy. Aby minimalizované ekvivalentní automaty vypadaly stejně, je nutné dodržet stejné pořadí vstupních symbolů.

Tuto problematiku lépe vysvětlím na předešlém příkladu minimalizovaného automatu *A*. Automat *A* reprezentujeme grafem a budeme upravovat při průchodu.



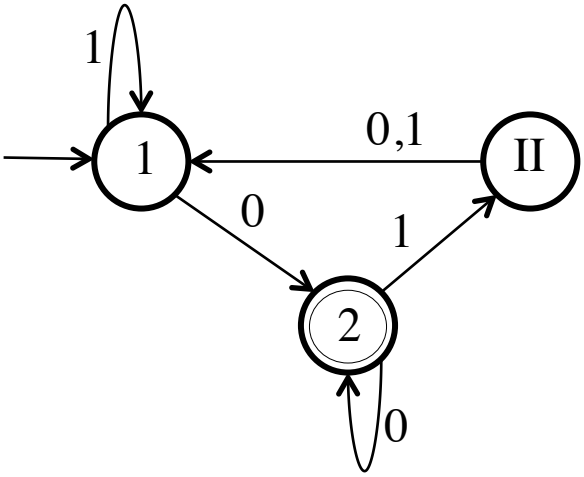
Obrázek 9: Automat *A* reprezentovaný grafem

Přepíšeme název počátečního stavu *I* na 1.



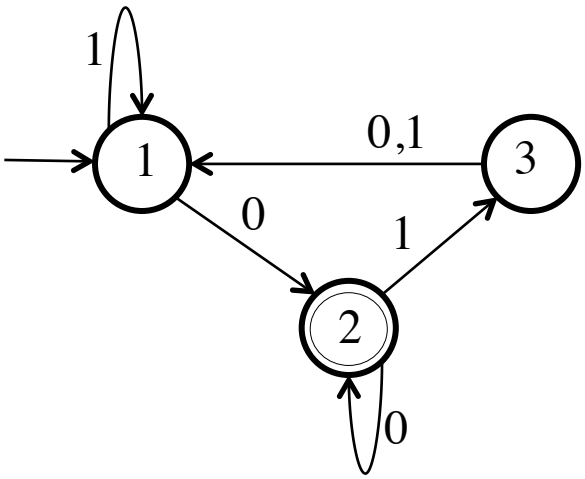
Obrázek 10: Upravený graf automatu *A* s přeznačeným prvním stavem

Pomocí přechodové funkce $\delta(1,0) = III$ se dostaneme na stav *III*, který označíme 2.



Obrázek 11: Upravený graf automatu *A* s přeznačeným druhým stavem

Nakonec označíme poslední neoznačený stav *II* jako stav 3.



Obrázek 12: Normovaný tvar automatu *A*

Nyní jsme přeznačili všechny stavy, čímž jsme získali automat v normovaném tvaru.

1.8 Převod konečného automatu na regulární výraz

Než začnu popisovat algoritmus převodu konečného automatu na regulární výraz, musím se zastavit a vysvětlit samotný pojem regulární jazyk a regulární výraz.

1.8.1 Regulární jazyk

Třídu regulárních jazyků $RJ(\Sigma)$ nad abecedou Σ nazýváme nejmenší třídu jazyků nad Σ, která je uzavřena na regulární operace. Regulárními operacemi myslíme operace sjednocení, násobení a iterace.

Pro třídu regulárních jazyků $RJ(\Sigma)$ nad abecedou Σ platí:

1. $\varnothing \in RJ(\Sigma)$ a $\{a\} \in RJ(\Sigma)$ pro každé $a \in \Sigma$.
2. Pro sjednocení platí: $L_1, L_2 \in RJ(\Sigma) \Rightarrow L_1 \cup L_2 \in RJ(\Sigma)$.
3. Pro násobení platí: $L_1, L_2 \in RJ(\Sigma) \Rightarrow L_1.L_2 \in RJ(\Sigma)$.
4. Pro iteraci platí: $L \in RJ(\Sigma) \Rightarrow L^* \in RJ(\Sigma)$.

Regulární jazyky jsou právě ty, které lze získat z elementárních jazyků pomocí regulárních operací. Elementární jazyk je buď prázdný jazyk nebo jazyk tvořený jedním slovem, které má pouze jeden symbol.

Jazyk obsahující prázdné slovo ε je také regulární, protože pro takový jazyk platí $\{\varepsilon\} = \varnothing^*$.

Každý regulární jazyk je možné definovat pomocí elementárních jazyků a předpisu, jak na tyto elementární jazyky použít regulární operace. Právě k tomuto účelu využíváme regulární výrazy.

1.8.2 Regulární výraz

Množinu regulárních výrazů nad abecedou $\Sigma = (a_1, a_2, ..., a_n)$ označujeme $RV(\Sigma)$ a definujeme jako nejmenší množinu slov v abecedě $\{a_1, ..., a_n, \varnothing, \varepsilon, +, ., *, (,)\}$, kde $a_1, ..., a_n, \varnothing, \varepsilon, +, ., *, (,)$ jsou symboly, které nepatří do abecedy, a platí:

1. Symboly $\varnothing$, ε a a nazveme elementární regulární výrazy, pro které platí $\varnothing \in RV(\Sigma)$, $\varepsilon \in RV(\Sigma)$ a $a \in RV(\Sigma)$ pro každý prvek a z abecedy Σ.
2. Pokud $\alpha, \beta \in RV(\Sigma)$ (jsou regulární výrazy), potom také $(\alpha + \beta) \in RV(\Sigma)$, $(\alpha.\beta) \in RV(\Sigma)$, $\alpha^* \in RV(\Sigma)$ a $\beta^* \in RV(\Sigma)$.
3. V množině $RV(\Sigma)$ nejsou žádné další výrazy, jsou tam pouze výrazy vytvořené z $\varnothing$, ε a prvků abecedy Σ pomocí pravidel uvedených v bodě 2.

Každý regulární výraz tvoří nějaký regulární jazyk, výraz $\varnothing$ označuje prázdný jazyk, ε jazyk $\{\varepsilon\}$, pro všechna $a \in \Sigma$ označuje a jazyk $\{a\}$.

Jsou-li α, β regulární výrazy, potom:

1. $(\alpha + \beta)$ označuje formální neuspořádaný součet, tedy na daném místě musí být buď výraz α, nebo výraz β.
2. $(\alpha.\beta)$ označuje zřetězení regulárních výrazů, to znamená, že zadané výrazy α a β po sobě bezprostředně následují. Zde je důležité zdůraznit, že $(\alpha.\beta)$ nemá stejný význam jako $(\beta.\alpha)$, jinými slovy záleží na pořadí výrazů.
3. α^* vyjadřuje iteraci, tedy regulární výraz α se může n-krát opakovat.

Pokud α označuje regulární jazyk L_1 a β jazyk L_2, potom $(\alpha + \beta)$ označuje $L_1 \cup L_2$, $(\alpha.\beta)$ označuje $L_1.L_2$, α^* označuje L_1^*. Jazyk reprezentovaný regulárním výrazem α obecně označujeme $[\alpha]$.

Pro větší přehlednost regulárních výrazů lze vynechat přebytečné závorky. Můžeme vynechat vnější pár závorek a další díky faktu, že součet i součin jsou asociativní. Pro regulární výrazy α, β, γ platí $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma) = \alpha + \beta + \gamma$ a také $(\alpha.\beta).\gamma = \alpha.(\beta.\gamma) = \alpha.\beta.\gamma$. Dále definujeme prioritu operací, díky které můžeme také některé závorky vypustit. Nejvyšší prioritu má iterace (*), následuje zřetězení (.) a nejnižší prioritu má součet (+).

Dalším zjednodušením, které se používá u operace zřetězení, je vynechání „.“. Tedy budeme zjednodušeně psát $\alpha\beta$ místo $\alpha.\beta$.

1.8.3 Regulární jazyk a konečný automat

Každý regulární jazyk je rozpoznatelný konečným automatem.

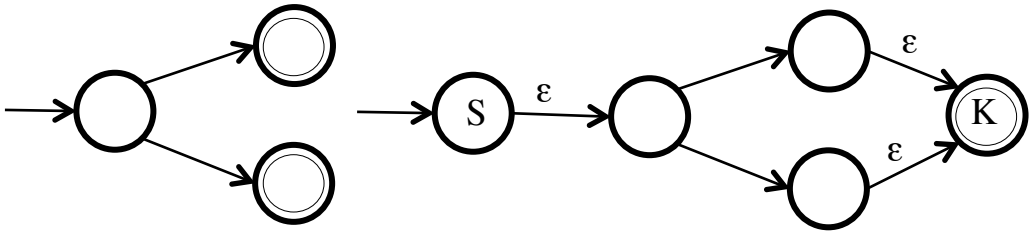
Je-li α regulární výraz, potom existuje konečný automat rozpoznávající jazyk $L(\alpha)$. Pro konečnou abecedu Σ lze sestrojit konečné automaty, které rozpoznávají elementární jazyky, tedy $\{\varnothing\}$, $\{\varepsilon\}$ a $\{a\}$ pro každé $a \in \Sigma$. Třída regulárních výrazů je uzavřená vůči sjednocení, zřetězení a iteraci. Z toho vyplývá, že každý regulární výraz α lze převést na zobecněný nedeterministický konečný automat A, který rozpoznává $[\alpha]$.

Každý jazyk rozpoznatelný konečným automatem je regulární. Pro každý konečný automat A existuje regulární výraz takový, že $[\alpha] = L(A)$.

1.8.4 Převod konečného deterministického automatu na regulární výraz

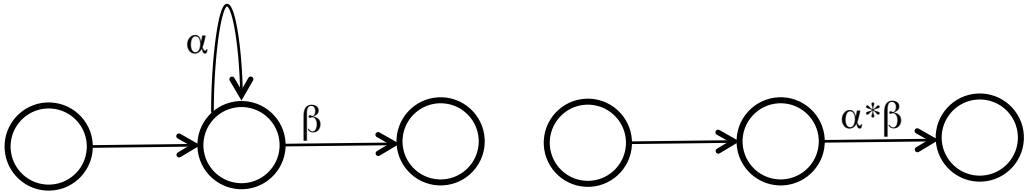
Máme konečný deterministický automat $A = (Q, \Sigma, \delta, q_0, F)$ a reprezentujeme jej pomocí grafu, který upravujeme. Celý algoritmus převodu automatu *A* na regulární výraz se skládá z následujících kroků:

1. Do grafu automatu přidáme dva nové stavy (vrcholy). Nový počáteční *S* a nový koncový stav *K*. Přidáme nové hrany. Nový počáteční stav *S* spojíme s původním počátečním stavem hranou a ohodnotíme ji prázdným slovem ε. Všechny původní rozpoznávací stavy spojíme hranou s novým koncovým stavem *K* a hrany ohodnotíme opět prázdným slovem ε. Automat má po prvním kroku jeden počáteční stav *S* a jeden koncový stav *K*. Jazyk, který takto upravený automat rozpoznává, se nezmění, protože automat může volně přecházet mezi stavy přes ε.



Obrázek 13: Přidání stavu *S* a stavu *K* do grafu

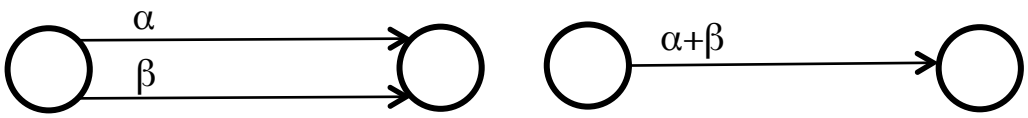
2. Projdeme automat a odstraníme všechny smyčky. Smyčka je hrana spojující stav (vrchol) sám se sebou. Po odstranění smyčky všechny hrany, které vychází ze stavu ohodnotíme regulárním výrazem, který vyjde zřetězením iterace výrazu na smyčce a výrazem, kterým byla ohodnocena hrana vycházející ze stavu. Jazyk rozpoznávaný automatem se nezmění, protože výraz α na smyčce se může opakovat, a pro změnu na další stav musí následovat výraz β. Po odstranění smyčky musí pro přechod na další stav přijít regulární výraz $\alpha^*\beta$, což podle definice iterace a zřetězení odpovídá automatu se smyčkou.



Obrázek 14: Odstranění smyčky z grafu

3. V dalším kroku odstraníme paralelní hrany. Jsou-li dva stavy spojeny více než jednou hranou, potom jsou právě tyto hrany paralelní. Hrany nahradíme

jedinou hranou, kterou ohodnotíme součtem výrazů, pomocí kterých byly ohodnoceny původní paralelní hrany. Pro paralelní hrany platí, že automat z jednoho stavu do druhého může přejít buď přes výraz α, nebo přes výraz β, čemuž podle definice součtu odpovídá regulárnímu výrazu $\alpha+\beta$, tedy jazyk rozpoznávaný automatem se nezmění.



Obrázek 15: Odstranění paralelních hran

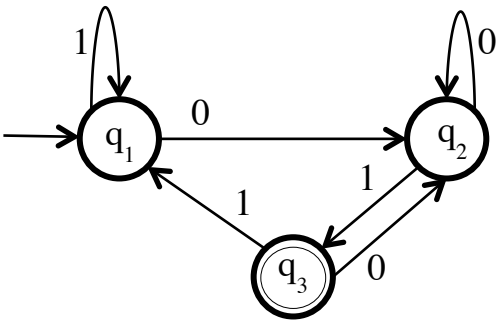
4. Dále odstraníme jeden stav z původních stavů automatu (neodstraňujeme ani nový počáteční *S* ani nový koncový *K*). Po odstranění stavu odstraníme všechny hrany, které do stavu vstupovaly i hrany, které ze stavu vycházely. Výrazy, kterými byly původní hrany ohodnoceny, zřetězíme a vytvoříme hranu novou s tímto ohodnocením. V tomto případě se opět nezmění jazyk rozpoznávaný automatem, protože aby automat přešel z prvního stavu přes druhý na třetí, musí přijít výraz α a potom β, což podle definice zřetězení odpovídá regulárnímu výrazu $\alpha\beta$ po odstranění druhého stavu.

Obrázek 16: Odstranění stavu z grafu

5. Opakujeme kroky 2, 3 a 4 dokud nám nezbude automat, který má pouze dva stavy, a to počáteční *S* a koncový *K*, mezi kterými je jen jedna hrana. Tato hrana je ohodnocena hledaným regulárním výrazem.

Pro osvětlení algoritmu předvedu celý postup na příkladu. Máme minimalizovaný konečný automat *A*, který rozpoznává všechna slova končící 01. Platí $A=(Q,\Sigma,\delta,q_0,F)$, kde $Q=\{q_1,q_2,q_3\}$, $\Sigma=\{0,1\}$, $q_0=q_1$, $F=\{q_3\}$, $\delta(q_1,0)=q_2$, $\delta(q_1,1)=q_1$, $\delta(q_2,0)=q_2$, $\delta(q_2,1)=q_3$, $\delta(q_3,0)=q_2$, $\delta(q_3,1)=q_1$.

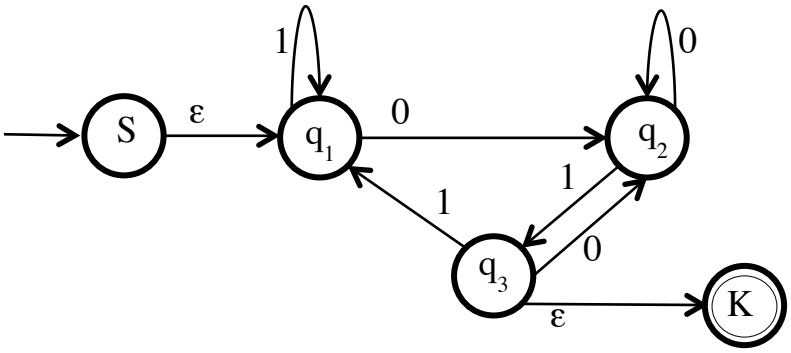
Zobrazíme automat pomocí grafu:



Obrázek 17: Graf automatu *A*

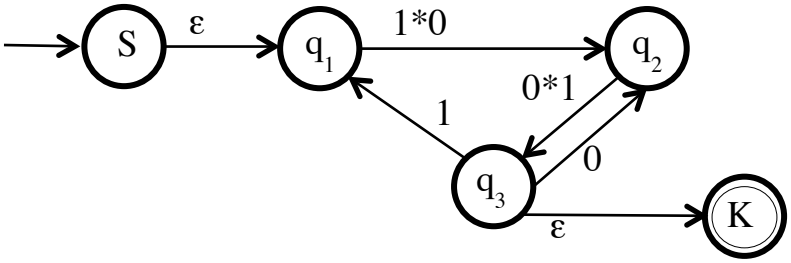
Dále budeme upravovat graf podle postupu převodu na regulární výraz:

1. Přidáme nový počáteční a koncový stav a upravíme hrany a jejich ohodnocení.



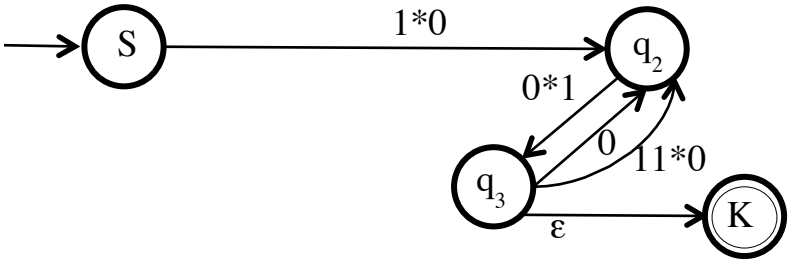
Obrázek 18: Přidání počátečního stavu *S* a koncového *K*

2. Odstraníme smyčky u stavů q_1 a q_2 a opět upravíme hrany a jejich ohodnocení.



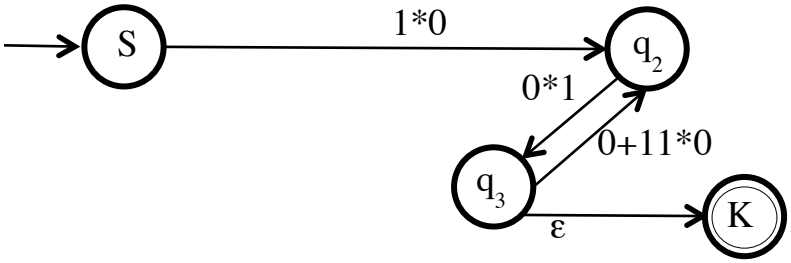
Obrázek 19: Odstranění smyček

3. Žádné paralelní hrany se v grafu nevyskytují. Přejdeme k dalšímu kroku, kde odstraníme stav q_1 a upravíme hrany a jejich ohodnocení.



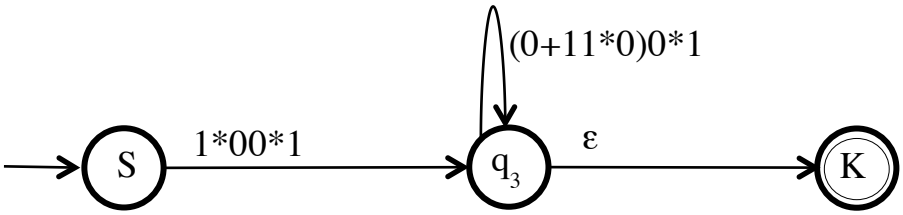
Obrázek 20: Odstranění stavu q_1

4. Automat má stále více než počáteční a koncový stav, proto pokračujeme. Při odstranění stavu vznikly mezi stavy q_2 a q_3 paralelní hrany, které sjednotíme a upravíme ohodnocení.



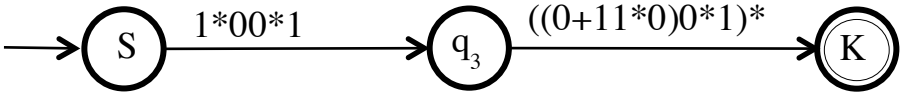
Obrázek 21: Odstranění paralelních hran

5. Odstraníme stav q_2 a upravíme hrany a jejich ohodnocení.



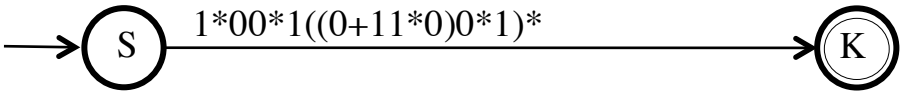
Obrázek 22: Odstranění stavu q_2

6. Odstraníme vzniklou smyčku u stavu q_3 a upravíme ohodnocení hrany vycházející z tohoto stavu.



Obrázek 23: Odstranění smyčky

7. Odstraníme stav q_3 a upravíme hrany a jejich ohodnocení.



Obrázek 24: Odstranění stavu q_3

8. Nyní jsme získali automat, který má pouze dva stavy – počáteční *S* a koncový *K*. Hledaným regulárním výrazem je ohodnocena poslední hrana. Výsledkem je regulární výraz $1*00*1((0+11*0)0*1)*$.

2 Praktická část

V této části se zaměřím na své vlastní řešení problému. Nejprve zmíním několik slov o technologiích, které jsem využívala, jimiž jsou programovací jazyk PHP, značkovací jazyk pro popis struktury automatu SCXML a další značkovací jazyk pro popis dvourozměrné vektorové grafiky SVG.

Dále se budu věnovat struktuře programu. Hlavní části jsou zadání a uložení konečného automatu, způsob vykreslení grafu zadaného automatu a zobrazení jednotlivých algoritmů. Algoritmy, které jsem v aplikaci zpracovávala, jsou převod nedeterministického automatu na deterministický, minimalizace konečného automatu a převod konečného automatu na regulární výraz.

V poslední kapitole se zaměřím na způsob, jakým provádím krokování algoritmů a na způsob vytváření animací algoritmů.

Fungování celé aplikace je možné si ověřit a vyzkoušet na http://kaja.nti.tul/~cernikova.

2.1 Použité technologie

Práci jsem vypracovala v jazyce PHP (jedná se o rekurzivní zkratku PHP: Hypertext Preprocessor, která původně znamenala Personal Home Page). Velkou výhodou programovacího jazyka PHP je jeho nezávislost na operačním systému. Pomocí tohoto skriptovacího jazyka jsem naprogramovala možnosti zadání konečného automatu, výpis informací o zadaném automatu a logiku všech předváděných algoritmů, včetně animace průběhu těchto algoritmů.

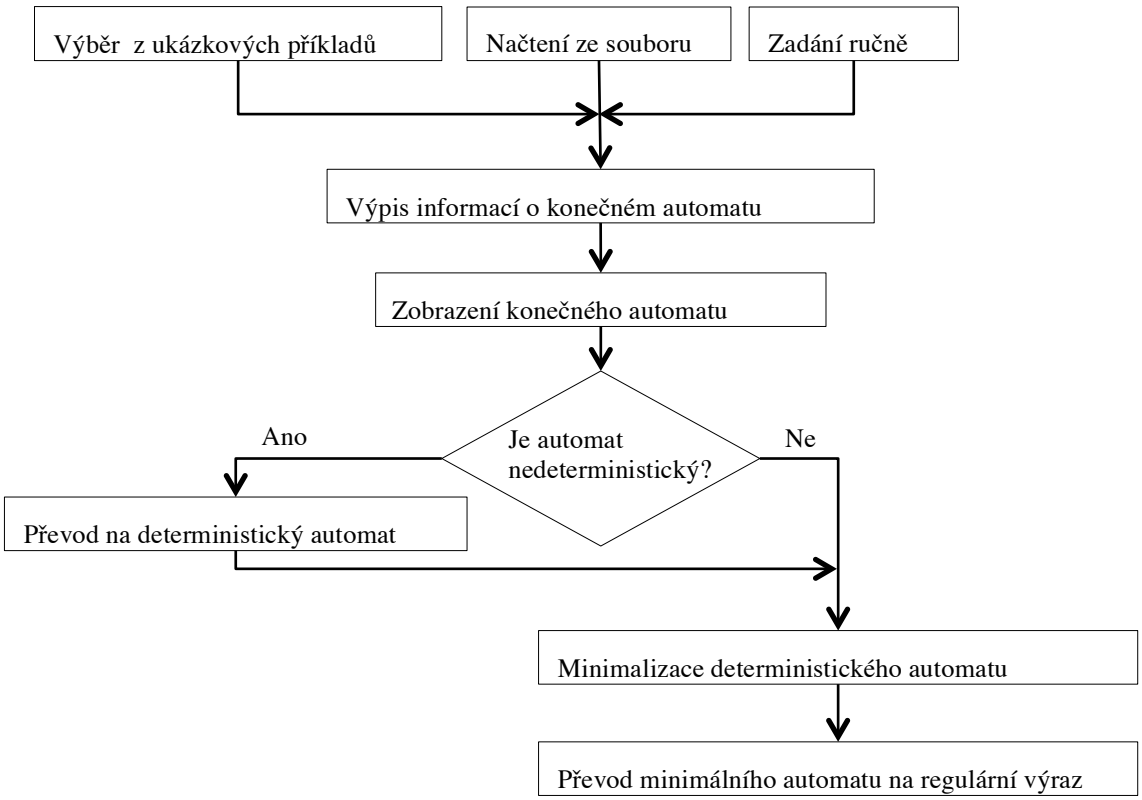
Pomocí SCXML (State Chart eXtensible Markup Language) jsem navrhla strukturu souboru pro uložení a načtení konečného automatu. Jazyk SCXML je určitým rozšířením značkovacího jazyka XML (eXtensible Markup Language). Vzhledem k tomu, že SCXML je vytvořené pro popis stavových diagramů, musela jsem schéma upravit, aby lépe vyhovovalo návrhu konečných automatů. Schéma, které jsem vytvořila pro nedeterministické konečné automaty, se nepatrně liší od schématu pro deterministické automaty, což vychází z rozdílů mezi jejich definicemi. Popisu jednotlivých schémat a jejich rozdílům se budu podrobně věnovat později v kapitole 2.4.

Dále pro mne byl důležitý značkovací jazyk SVG (Scalable Vector Graphics), který umožňuje popsat dvourozměrnou vektorovou grafiku. Opět se jedná o technologii, která je postavena na základu značkovacího jazyka XML. Použila jsem ji pro vykreslení

konečného automatu. Výhodou jazyka SVG je fakt, že zdrojovým kódem, který reprezentuje určitý obrázek, je prostý text. Díky textovému formátu se i minimalizují nároky na paměťový prostor, který takový obrázek zabírá. S jednotlivými objekty se dá jednoduše pracovat. Lze je seskupovat a proto i snadno upravovat. Další výhoda plyne z faktu, že SVG slouží pro vektorovou grafiku, kde se objekty nezapisují pomocí hodnot barevných pixelů uspořádaných do pravoúhlé mřížky, jak je tomu u rastrové grafiky, ale celý obrázek je složen ze základních geometrických objektů (body, přímky, křivky). A tak je možné obrázek vytvořený pomocí SVG zmenšovat a zvětšovat bez ztráty kvality, jako tomu je právě u rastrové grafiky.

2.2 Struktura aplikace

Celé řešení se skládá ze tří hlavních částí. Jsou jimi zadání konečného automatu, vykreslení grafu zadaného konečného automatu a reprezentace algoritmů převodu nedeterministického automatu na deterministický, minimalizace konečného automatu a převod konečného automatu na regulární výraz. Celá struktura je přehledně zobrazena na obrázku 25.



Obrázek 25: Schéma aplikace

Prvním krokem aplikace je zadání konečného automatu. Uživatel má možnost vybrat si z ukázkových příkladů, které jsem připravila, dále může načíst soubor, který má již připravený, nebo zadat automat ručně pomocí formuláře.

Po zadání všech údajů, nebo jejich načtení ze souboru dojde k jejich kontrole a nevyskytují-li se v údajích chyby, vypíší se všechny informace o zadaném automatu pomocí přechodové tabulky. Zde má uživatel možnost vše překontrolovat a eventuálně se vrátit a údaje opravit, také má možnost si automat uložit.

Dalším krokem je vykreslení automatu pomocí grafu.

V případě, že uživatel zadal nedeterministický automat, potom má možnost prohlédnout si algoritmus jeho převodu na deterministický.

Po získání deterministického automatu nebo zadal-li uživatel rovnou deterministický automat, má možnost zobrazit algoritmus jeho minimalizace.

Posledním algoritmem, který má aplikace prezentuje, je převod minimalizovaného konečného automatu na regulární výraz.

Všechny algoritmy si uživatel může odkrokovat. Má také možnost spustit algoritmus automaticky jako animaci. Potom se nemusí proklikávat přes všechny kroky postupu, což je výhodné, pokud má algoritmus větší množství kroků.

V dalších kapitolách se podrobně zaměřím na jednotlivé části aplikace a jejich zpracování.

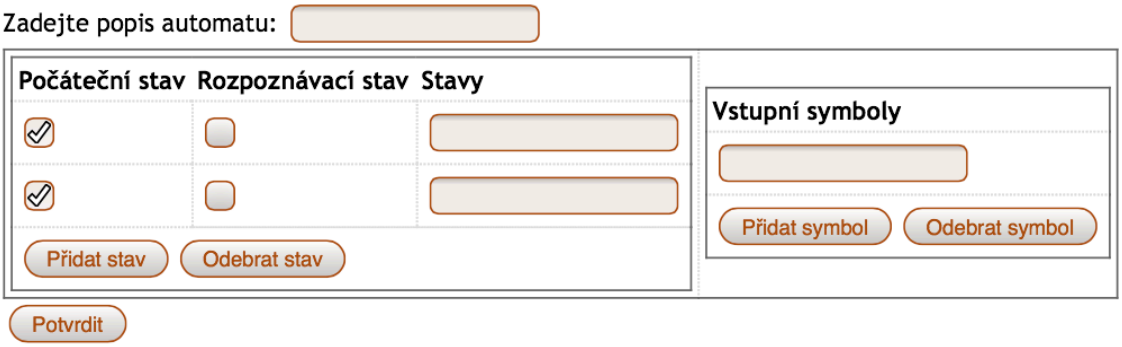
2.3 Zadání konečného automatu

Jak jsem již zmínila v předchozí kapitole, konečný automat je možné zadat třemi způsoby: z ukázkových příkladů, ze souboru a ručně. Tyto tři možnosti jsem zvolila, protože podle mne pokrývají potřeby uživatelů při zadávání konečného automatu.

Ukázkové příklady jsem vytvořila, aby si uživatelé mohli vyzkoušet, jak aplikace funguje, a nemuseli vymýšlet vlastní automat. Připravila jsem příklady pro nedeterministické i pro deterministické automaty. Automaty rozpoznávají různé jazyky. Uživatelé mají možnost, prohlédnout si schémata ukázkových automatů, což jim může pomoc při vytváření jejich vlastních souborů.

Pokud již uživatel vytvořil nějaký automat, potom má možnost ho načíst a dále s ním pracovat. Možnost načtení automatu ze souboru je výhodné v případě, že uživatel chce zadat složitější automat, jehož zadání ručně pomocí formuláře by bylo náročné a nepřehledné. V případě, že uživatel soubor nahraje, potom proběhne jeho validace. Pokud je soubor validní, tedy má správnou strukturu (strukturu souboru rozebírám podrobně v kapitole 2.4), potom kontroluji formální správnost údajů. Všechny vstupní symboly a stavy musí mít unikátní názvy. Nesmí obsahovat zakázané znaky (povoluji pouze kombinaci písmene a čísel), musí být zadán počáteční a alespoň jeden koncový stav.

Poslední možností je zadat automat ručně. Ruční zadání je výhodné, chce-li uživatel vytvořit jednodušší automat. Uživatel musí nejprve vybrat, zda chce vytvořit automat nedeterministický nebo deterministický. Pro oddělení zadání těchto dvou typů automatů jsem se rozhodla s ohledem na jejich rozdílnou definici. Po vybrání jedné z možností se zobrazí formulář pro zadání údajů o automatu. Formulář pro zadání nedeterministického automatu zobrazuje obrázek 26.



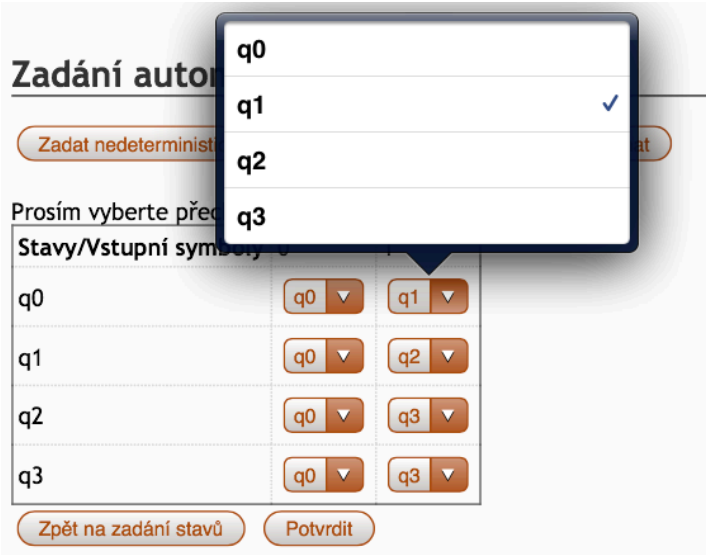
Obrázek 26: Zadání nedeterministického automatu

Uživatel musí zadat popis automatu, názvy jeho stavů a vstupních symbolů. Dále musí určit, které stavy jsou počáteční a které stavy jsou rozpoznávací.

Pro zadání deterministického konečného automatu vypadá tento formulář stejně až na jeden rozdíl a to ve výběru počátečních stavů. Tento rozdíl vyplývá z rozdílu definice deterministického a nedeterministického automatu. Konkrétně se to týká faktu, že nedeterministický automat může mít více než jeden počáteční stav (přesněji má nedeterministický automat množinu počátečních stavů $I \subseteq Q$) a deterministický automat má počáteční stav q_0 právě jeden. Proto u deterministického automatu povoluji vybrání jen jednoho počátečního stavu.

Ve chvíli, kdy uživatel potvrdí zadané údaje, proběhne kontrola správnosti údajů. Názvy stavů a vstupních symbolů musí být unikátní a musí se skládat pouze z písmen a číslic. Dalším omezením je, že názvy vstupních symbolů mohou být tvořeny pouze jedním písmenem, nebo jednou číslicí. Tato omezení jsem zvolila, protože se nedotýkají funkčnosti aplikace. Takto zvolené názvy jsou pro konečné automaty dostatečné a vyhnu se zadání znaků, které by mohly způsobit nestandardní situace. Dále musí být vybrán alespoň jeden stav jako počáteční (u nedeterministických automatů alespoň jeden, u deterministických právě jeden) a alespoň jeden stav jako rozpoznávací. Také nepovoluji uživateli ponechat názvy stavů nebo symbolů prázdné. Vyskytnou-li se při zadání chyby, potom uživatele upozorním tím, že vypíši, kde a jaké chyby udělal a nedovolím mu pokračovat, dokud chyby neopraví.

V případě, že se během kontroly neobjevily chyby, potom se uživateli zobrazí formulář pro zadání přechodových funkcí.



Obrázek 27: Výběr přechodových funkcí u deterministického automatu

Obrázek 28: Výběr přechodových funkcí u nedeterministického automatu

Jak vidíme na obrázku 27, uživatel má před sebou v tuto chvíli tabulku, která je podobná přechodové tabulce. V tabulce jsou v prvním řádku vyplněné vstupní symboly a v prvním sloupci jsou stavy automatu. Uživatel musí vybrat, do jakého stavu automat přejde přes konkrétní vstupní symbol. Toto řešení jsem zvolila hlavně kvůli přehlednosti. Také jsem zkoušela možnost, že by uživatel vyplňoval přechodové funkce ručně, ale to by mohlo vést k zbytečným chybám. Také bych u každého zadání musela kontrolovat, zda uživatel zadal pouze stavy, které se v množině stavů automatu vyskytují. Proto jsem se rozhodla, že uživatel bude vybírat pouze z předem připravených možností.

Mezi formuláři pro zadání přechodové funkce je samozřejmě rozdíl mezi deterministickým (obrázek 27) a nedeterministickým (obrázek 28) automatem. Nedeterministický automat může přejít z jednoho stavu přes vstupní symbol do množiny stavů (tedy podle definice přechodové funkce $Q \times \Sigma \rightarrow P(Q)$, kde $P(Q)$ je množina všech možných podmnožin Q), proto má uživatel na výběr ze všech kombinací stavů. V případě, že přechodová funkce pro konkrétní stav a vstupní symbol nemá být definována, uživatel vybere symbol „-“. U deterministického automatu přechází automat vždy z jednoho stavu přes vstupní symbol pouze na jeden stav (tedy podle definice $Q \times \Sigma \rightarrow Q$), uživatel má tedy možnost vybrat pouze jeden cílový stav.

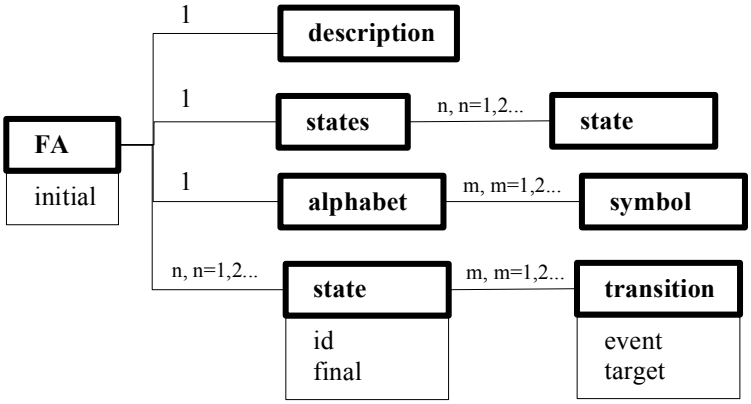
Po vybrání přechodových funkcí může uživatel přejít na výpis informací o automatu, který zadal, a následně k jeho vykreslení.

Může se také vrátit na předchozí formulář. Tato možnost může být výhodná hlavně v případě, že chce přidat další stav nebo vstupní symbol nebo chce změnit jejich názvy.

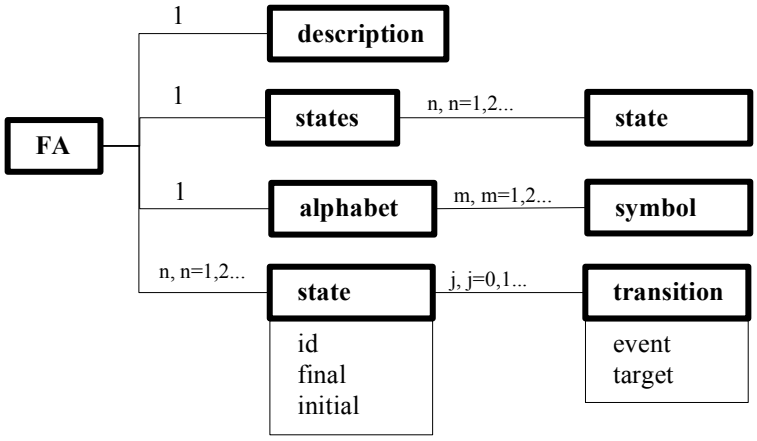
2.4 Uložení automatu pomocí XML souboru

Při vytváření struktury souboru, pomocí kterého je možné automat načíst nebo uložit jsem zvažovala několik možností. První byla navrhnout vlastní strukturu. Tuto možnost jsem nakonec zavrhla, protože již existují řešení, jak uložit stavové automaty pomocí schématu vytvořeném v jazyce SCXML (State Chart XML), které stačí upravit tak, aby vyhovovaly konečným automatům. SCXML je jazyk vycházející z XML (eXtensible Markup Language) a pro uživatele aplikace je jednodušší naučit se XML schéma, než se snažit zorientovat ve struktuře, která by byla úplně nová nebo popsaná v novém jazyce.

Jak jsem již naznačila, při vytváření struktury souboru pro uložení automatů jsem vycházela z jazyka SCXML. Vzhledem k definici deterministických automatů mi ovšem chyběla možnost přesně ve schématu definovat všechny stavy automatu a abecedu. Také jsem z důvodu praktičnosti přidala element pro popis (description) automatu. Pro nedeterministické automaty jsem musela udělat více úprav tak, aby bylo možné pro každý stav definovat, zda je počáteční nebo není. U deterministického automatu se počáteční stav definoval v kořenovém elementu, což je pro nedeterministické automaty nevýhodné.



Obrázek 29: Výsledné schéma XML deterministického automatu



Obrázek 30: Výsledné schéma XML nedeterministického automatu

Na obrázcích 29 a 30 jsou znázorněna upravená schémata pro deterministické a nedeterministické automaty. Kořenovým elementem je element FA (tedy Finite Automata, neboli konečný automat). U deterministických automatu má tento element atribut initial, který označuje počáteční stav. U nedeterministického automatu tento atribut v elementu FA není, protože takový automat může mít více počátečních stavů. V kořenovém elementu jsou vnořeny elementy description, states, alphabet, a state.

Element description v sobě skrývá popis automatu, tento element se musí vyskytovat na tomto místě právě jednou.

Dalším elementem, který musí být právě jeden, je states, který reprezentuje neprázdnou množinu stavů konečného automatu a musí tedy obsahovat minimálně jeden element state. Elementy state, které jsou vnořené do elementu states, obsahují názvy stavů konečného automatu.

Element alphabet je pro abecedu konečného automatu a obsahuje minimálně jeden element symbol, který reprezentuje vstupní symbol abecedy.

Posledním elementem, který je vnořený do kořenového elementu FA, je state. Element state má atributy id, final a pro nedeterministický automat navíc atribut initial. Id označuje název stavu. Pokud má atribut final hodnotu true, potom je stav rozpoznávací, jinak má hodnotu false. U nedeterministických automatů má atribut initial hodnotu true, pokud je stav počáteční, jinak má hodnotu false.

Do elementu state jsou vnořeny elementy transition, které reprezentují přechodovou funkci. U deterministických automatů musí mít každý element state vnořeno tolik elementů transition, kolik je vstupních symbolů. To vyplývá z definice deterministických automatů. Přechodová funkce deterministického automatu je úplné zobrazení kartézského součinu $Q \times \Sigma$ do množiny Q. Na rozdíl od deterministického je

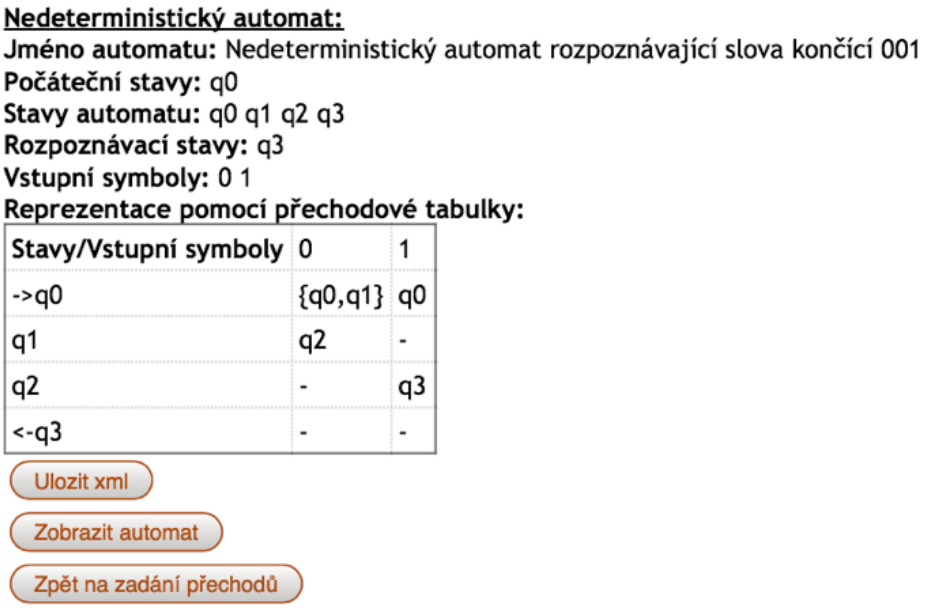
přechodová funkce nedeterministického automatu definována jako zobrazení kartézského součinu $Q \times \Sigma$ do množiny podmnožin Q, tedy z jednoho stavu je možné přejít do množiny stavů. Navíc nemusí být přechodová funkce definována pro všechny dvojice (q,a), kde $q \in Q$ a $a \in \Sigma$. Právě proto u nedeterministických automatů nemusí mít element state vnořený element transition. Element transition má právě dva atributy, jimiž jsou event a target. Event reprezentuje vstupní symbol a target stav, do kterého automat přejde po příchodu vstupního symbolu označeného v atributu event.

Vytvořila jsem RelaxNG schéma pro XML soubor. Toto schéma využívám pro validaci, tedy ověření zda má zadaný XML soubor správné schéma.

Abych uživatelům co nejvíce usnadnila práci se svou aplikací, vytvořila jsem dvě šablony (pro deterministický a nedeterministický automat), které si uživatelé mohou stáhnout z hlavní stránky aplikace. Šablony obsahují XML pro zadání automatů pomocí souboru a jsou doplněny podrobným popisem, aby uživatel věděl, jak je má vyplnit.

2.5 Výpis informací o zadaném automatu

Nezávisle na způsobu, jakým uživatel zadal konečný automat, vypadá výpis informací o automatu vždy stejně. Jedná se o rekapitulaci zadaných údajů, než přikročíme k vykreslení grafu automatu. Samozřejmě, pokud uživatel nyní zjistil, že při zadání udělal chybu, může se vrátit a chybu napravit.



Obrázek 31: Výpis informací o zadaném automatu

Na obrázku 31 je zobrazen příklad vypsání informací o nedeterministickém automatu. V tomto případě se jedná o automat rozpoznávající slova končící 001.

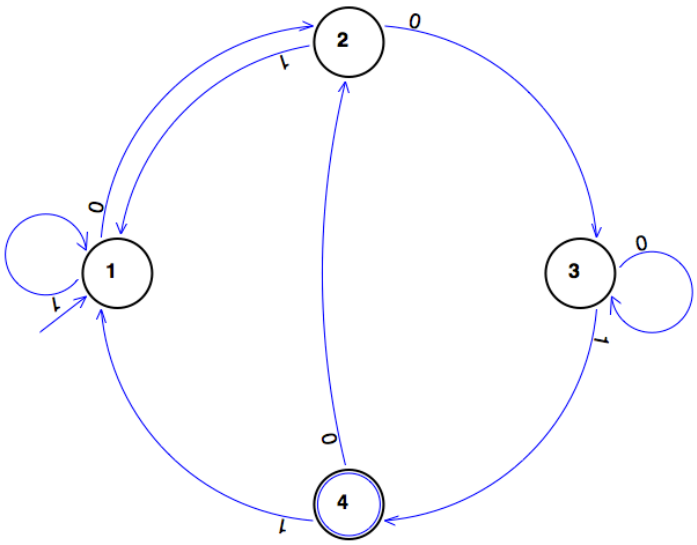
Uživatel má na tomto místě možnost uložit zadaný automat. V takovém případě se vygeneruje XML podle zadaných údajů o automatu a soubor se uloží na místo, které uživatel vybere (záleží na prohlížeči, který uživatel používá, některé prohlížeče stahují soubory automaticky do předem nastavené složky).

Tlačítko „Zpět na zadání přechodů“ vrátí uživatele na formulář pro zadání přechodových funkcí. Aby uživatel nemusel při návratu vybírat všechny přechody znovu, tak formulář pro zadání přechodové funkce vyplňuji údaji, které již zadal. Díky tomu se uživatel může zaměřit jen opravu přechodů, které mu nevyhovují.

Poslední možností je zobrazení automatu, díky které uživatel přejde na stránku, na které se vykreslí graf zadaného automatu. Tam v závislosti na typu zadaného konečného automatu bude mít uživatel možnost pokračovat. Pro nedeterministický automat si lze zobrazit jeho převod na deterministický automat, nebo pro deterministický automat si lze projít algoritmus jeho minimalizace.

2.6 Vykreslení grafu automatu

Graf automatu vykresluji pomocí značkovacího jazyka SVG (Scalable Vector Graphics), který slouží k popisu dvourozměrné vektorové grafiky. SVG generuji pomocí jazyka PHP. Na obrázku 32 je vidět zobrazení grafu deterministického konečného automatu, který rozpoznává všechna slova končící 001.



Obrázek 32: Zobrazení deterministického automatu

Stavy automatu zobrazuji jako černé kružnice. Uvnitř kružnice stavu je název stavu. Počáteční stav je označen modrou šipkou, která vstupuje do kružnice. Koncový stav je označen menší modrou kružnicí, která je uvnitř kružnice stavu. Pro zobrazení přechodových funkcí jsem zvolila modré šipky směřující od jednoho stavu k druhému. Každá šipka je ohodnocena vstupním symbolem. Podle obrázku 32 je zřejmé, že se vstupní symboly natáčí podle orientace šipky, která znázorňuje přechodovou funkci. Samozřejmě jsem zkoušela i variantu, že vstupní symbol bude vždy natočen „rovně“. Protože je možné, že ne všem vyhovuje řešení natáčení vstupních symbolů, protože některé vstupní symboly jsou úplně vzhůru nohama. Když jsem ale zkoušela vytvořit graf s větším množstvím přechodů, jejichž šipky se křížily, tak se mi varianta otáčení zdála lepší, protože bylo lépe čitelné, který vstupní symbol patří ke které šipce, a vykreslení grafu tak bylo přehlednější.

Kružnice stavů mají pevný poloměr, což přináší omezení v délce názvu stavu. Dlouhé názvy se do kružnice nevejdou a přesahují ji, což neomezuje funkčnost, ale samozřejmě při použití dlouhého názvu stavu se může stát graf nepřehledný. To ale jen do chvíle, než automat projde prvním algoritmem (převod na deterministický automat

nebo minimalizace), protože tam provádím jeho převod do normovaného tvaru a proto názvy stavů přepíši čísly $1,...,n$.

Jednotlivé kružnice stavů rozmísťuji na kružnici, jejíž poloměr měním podle počtu stavů. Pro výpočet souřadnic středů kružnic stavů (x_S, y_S) využívám parametrické rovnice kružnice podle definice Dubcové, Purmové a Simerské (*citace 3*): $x = m + r\cos t$ a $y = n + r\sin t$, kde bod $S = [m,n]$ je její střed, r je poloměr, x, y jsou souřadnice libovolného bodu na kružnici a $t \in R$ je parametr, který udává úhel mezi úsečkou spojující střed kružnice k bodem na kružnici a kladný směr osy x. Platí, že $0 \leq t \leq 2\Pi$. Abych tyto vzorce mohla použít pro vykreslení stavů, musela jsem ho upravit a to následovně:

1. pro souřadnici x_S vzorec: $x_S = x_K + (r_K * \cos(\Pi + (2 * \frac{\Pi}{n}) * k))$. (9)

2. pro souřadnici y_S vzorec: $y_S = y_K + (r_K * \sin(\Pi + (2 * \frac{\Pi}{n}) * k))$. (10)

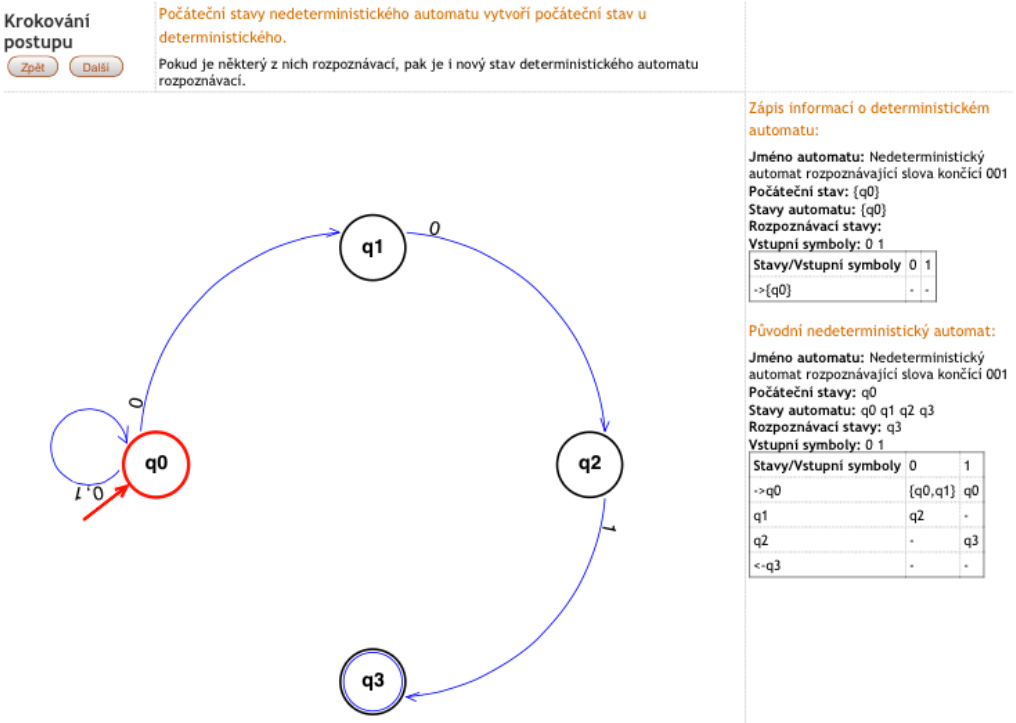
Hodnoty (x_K, y_K) jsou souřadnice středu kružnice, na jejímž obvodě se kružnice stavů nachází, střed této kružnice je vždy ve středu kreslící plochy a r_K je její poloměr.

Poloměr r_K se mění podle počtu stavů. Platí čím více stavů, tím větší tento poloměr je. Respektive podle počtu stavů upravuji rozměry kreslící plochy a podle té počítám poloměr r_K.

Vzorec $\Pi + (2 * \frac{\Pi}{n}) * k$ slouží pro výpočet úhlu, který se mění v závislosti na počtu stavů n automatu a počtu již vykreslených stavů k. Tedy podrobněji: $2 * \frac{\Pi}{n}$ je úhel, který svírá spojnice středu kružnice stavu s osou x. Tento úhel násobím k, což je počet vykreslených stavů, čímž dostávám postupně všechny úhly, které svírají spojnice středů kružnic stavů s osou x. Nakonec přičítám Π, abych začala vykreslovat stavy úplně vlevo, a potom postupuji dále po směru hodinových ručiček.

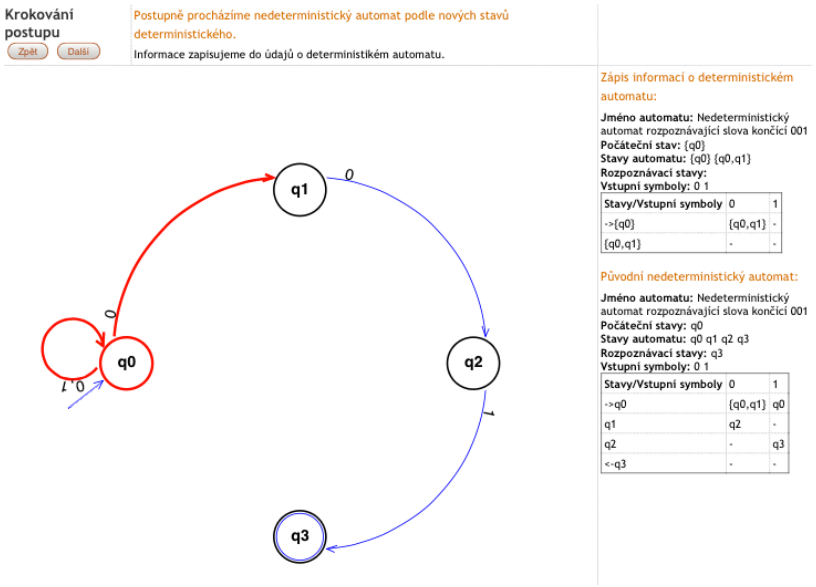
2.7 Převod nedeterministického automatu na deterministický

Algoritmus pro převod nedeterministického automatu na deterministický jsem podrobně popsala v kapitole 1.5. Proto nyní popíši jen své ztvárnění tohoto algoritmu na příkladu nedeterministického automatu, který rozpoznává slova končící 001. Výsledkem bude deterministický konečný automat rozpoznávající stejná slova, tedy končící 001.

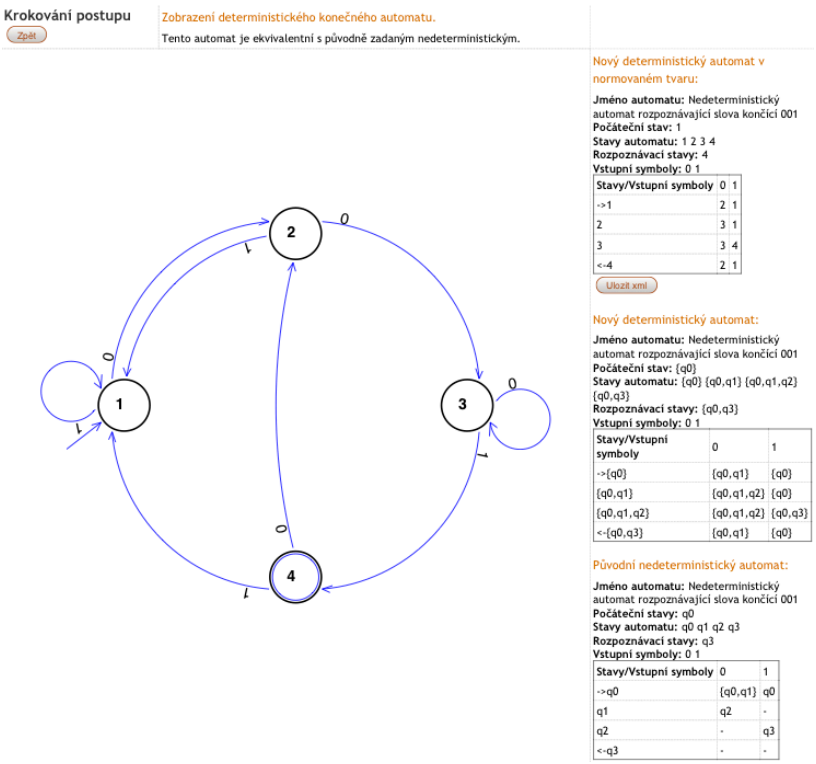


Obrázek 33: První krok převodu na deterministický automat

Na obrázku 33 je vidět první krok převodu na deterministický konečný automat při využití možnosti krokování algoritmu. V horní části okna jsou tlačítka pro krokování algoritmu, zpět a další. Také je zde popis jednotlivých kroků. Napravo se nachází informace o původním nedeterministickém automatu, rovněž se tam zapisují nově získané údaje o novém deterministickém automatu. V hlavní části okna se vykresluje graf automatu, který se právě zpracovává. Při průchodu automatem označuji červeně právě zpracovávanou množinu stavů a zpracovávané hrany, jak je znázorněno na obrázku 34.



Obrázek 34: Krok převodu na deterministický automat

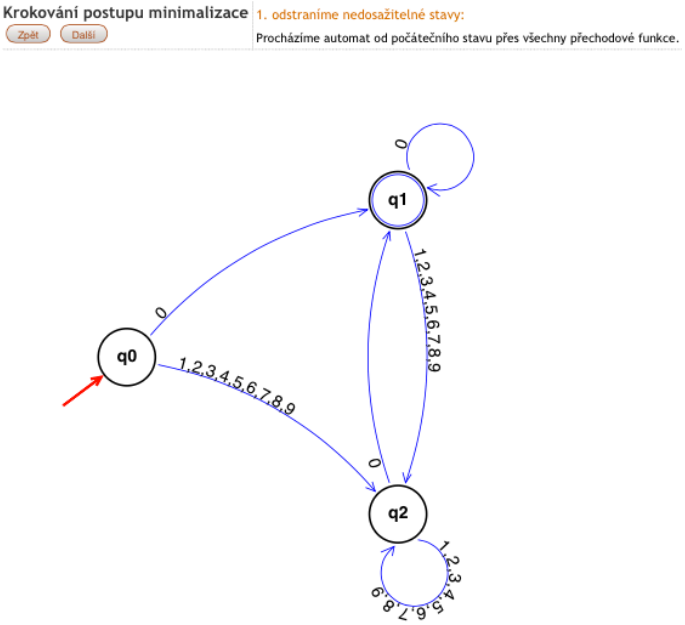


Obrázek 35: Poslední krok převodu na deterministický automat

Poslední okno algoritmu (obrázek 35) obsahuje převedený deterministický automat v normovaném tvaru, obsahuje informace o původním nedeterministickém automatu (ve formě přechodové tabulky) a výpis informací o deterministickém automatu v podobě množin stavů, tak jak vyšly po skončení algoritmu a také výpis toho automatu v normovaném tvaru. Uživatel má také možnost uložit si převedený deterministický automat. Má tedy možnost později si automat načíst a dále s ním pracovat.

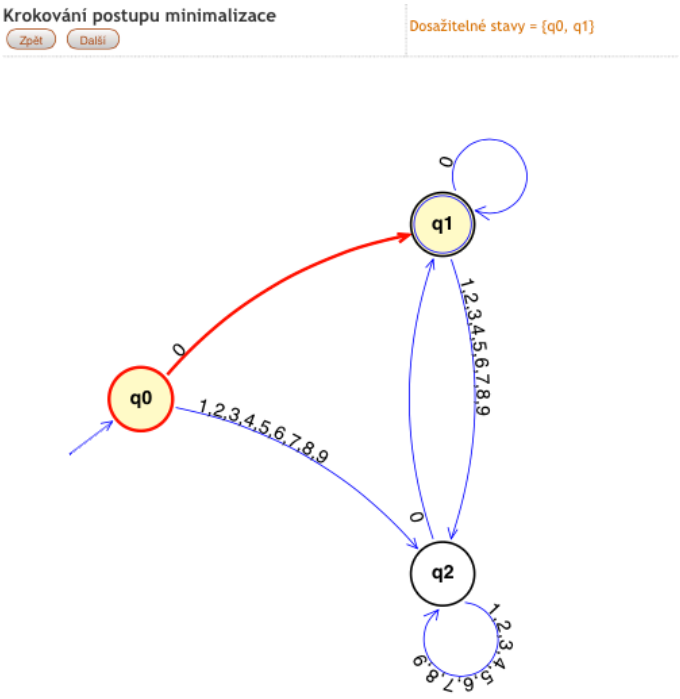
2.8 Minimalizace konečného automatu

Algoritmus minimalizace deterministického konečného automatu jsem popsala podrobně v kapitole 1.6. Algoritmus nyní předvedu ve zkratce na příkladu automatu rozpoznávajícího (přijímajícího) čísla dělitelná 10. Algoritmus minimalizace konečného automatu se skládá ze dvou částí. V první dochází k nalezení nedosažitelných stavů a jejich následném odstranění. V druhé části jsou nalezeny ekvivalentní stavy a potom sloučeny.

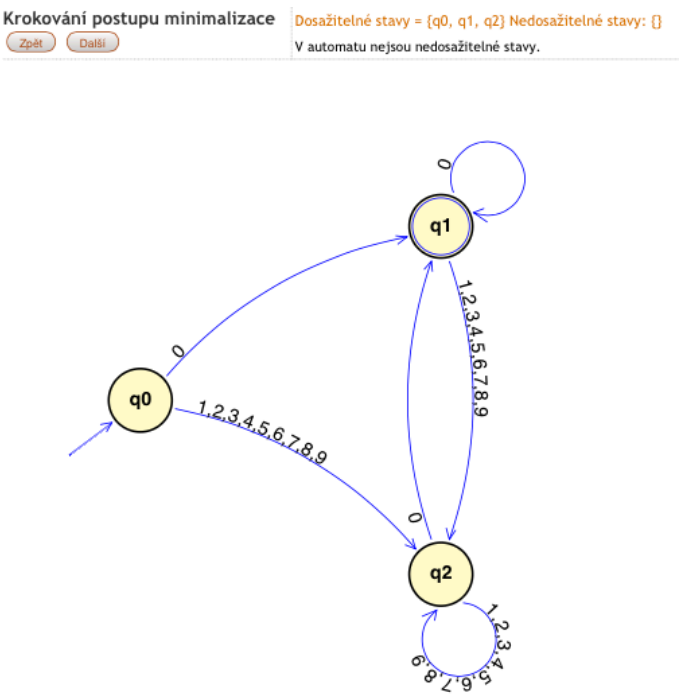


Obrázek 36: První krok odstranění nedosažitelných stavů

Na obrázku 36 je vidět první okno procesu odstranění nedosažitelných stavů. V horní části se opět nachází tlačítka pro krokování a popis postupu. V hlavní části je vykreslený graf automatu, který postupně procházíme od počátečního stavu přes přechodové funkce. Stavy, na které se dostaneme označíme jako dosažitelné a tyto stavy(vrcholy grafu) obarvíme žlutou barvou, aby bylo z obrázku jasně zřetelné, které stavy to jsou. Právě zpracovávaný stav je označen červeně. Rovněž právě zkoumaná přechodová funkce ve formě hrany grafu, je označena červeně, jak je vidět na obrázku 37.

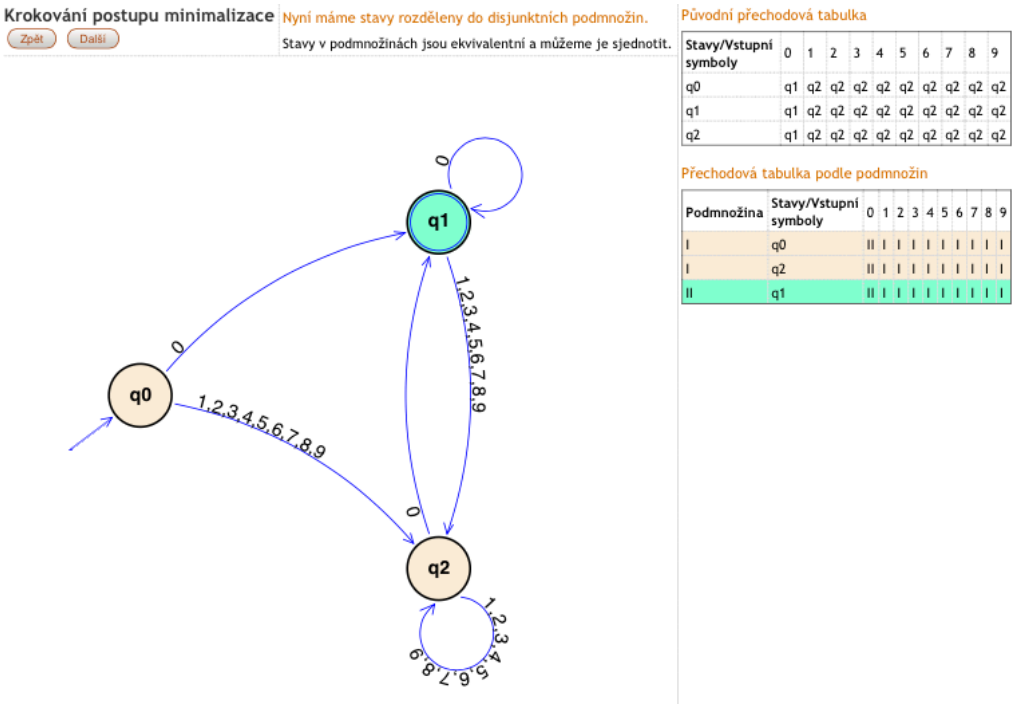


Obrázek 37: Krok odstranění nedosažitelných stavů



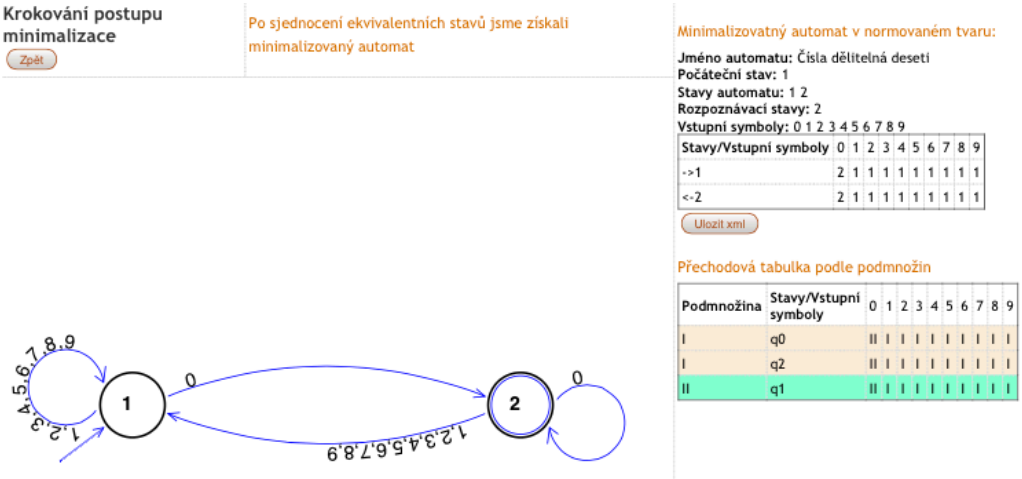
Obrázek 38: Poslední krok odstranění nedosažitelných stavů

Na obrázku 38 je znázorněn poslední krok hledání nedosažitelných stavů. V tomto případě byly všechny stavy dosažitelné, a tak nedošlo k odstranění žádného z nich. Dále přistupujeme k druhé části algoritmu, tedy k hledání ekvivalentních stavů a jejich následnému sloučení.



Obrázek 39: Krok sjednocení ekvivalentních stavů

Na obrázku 39 je jeden z kroků algoritmu. Vidíme rozdělení stavů do podmnožin podle rozpoznávacích stavů a obarvení grafu podle těchto podmnožin. Vykreslení grafu není nutné u hledání ekvivalentních stavů, protože celý proces stačí znázornit pomocí tabulky, ale díky zobrazení na grafu je celý algoritmus názornější, což se od výukové aplikace očekává.

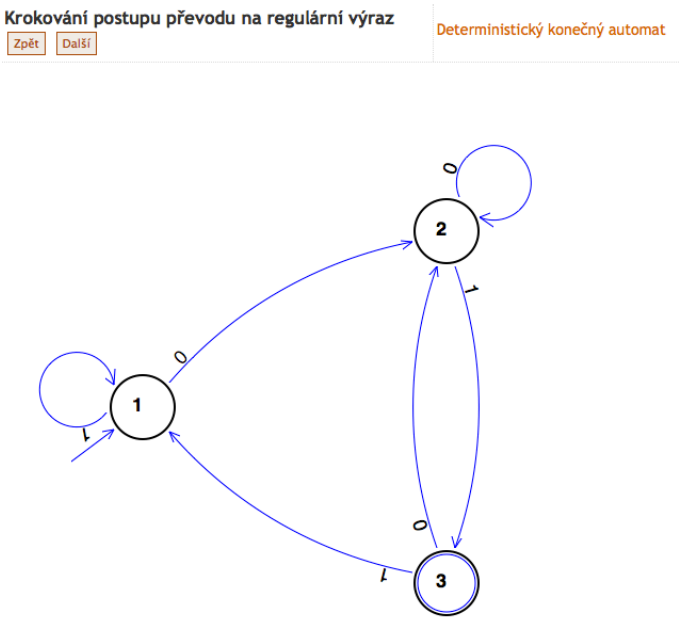


Obrázek 40: Poslední krok sjednocení ekvivalentních stavů

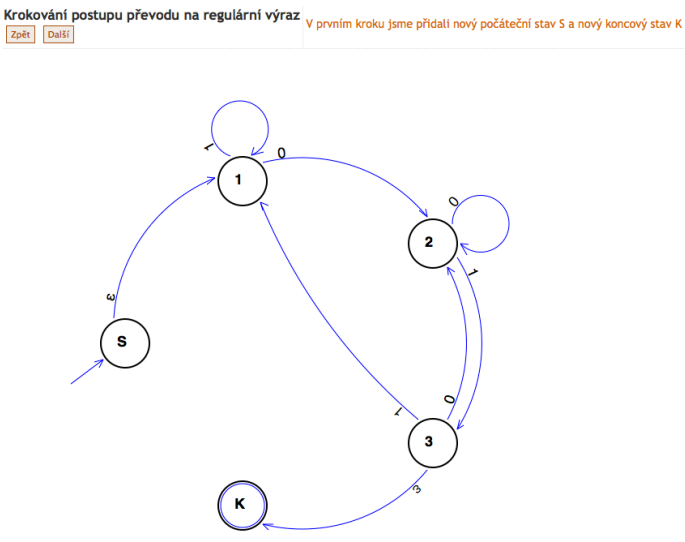
Na obrázku 40 je vidět výsledek algoritmu minimalizace. V pravé části obrazovky jsou dvě tabulky. První zobrazuje přechodovou tabulku minimalizovaného automatu a druhá ukazuje přechodovou tabulku původního automatu, kde jsou graficky odděleny množiny s ekvivalentními stavy, které jsme sloučili.

2.9 Převod konečného automatu na regulární výraz

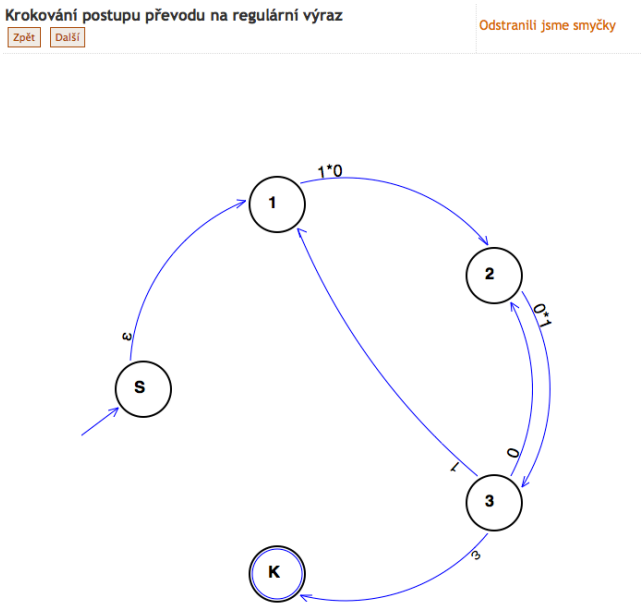
Algoritmus převodu deterministického konečného automatu na regulární výraz jsem již teoreticky popsala v kapitole 1.8. Nyní předvedu na konečném deterministickém automatu, který rozpoznává slova končící 01, své vlastní provedení algoritmu. Na rozdíl od ostatních algoritmů, kdy jsem přidávala jen ukázky prvního a posledního kroku, tento algoritmus ukáži celý. Výsledkem bude regulární výraz, který bude odpovídat slovům končícím 01.



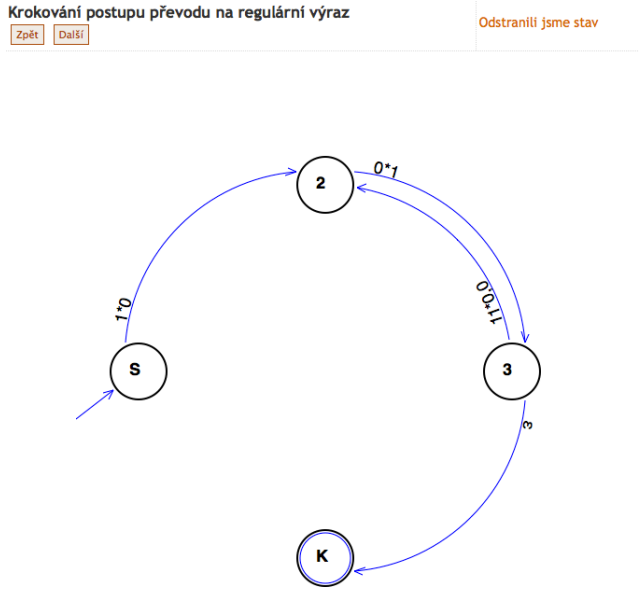
Obrázek 41: 1. krok převod na regulární výraz



Obrázek 42: 2. krok převod na regulární výraz

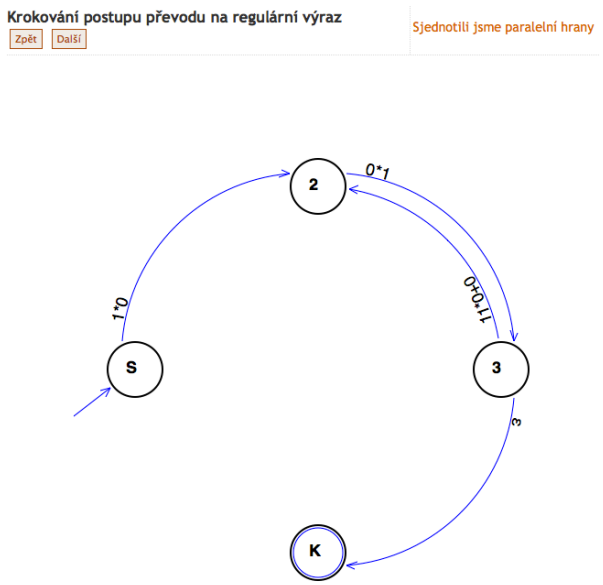


Obrázek 43: 3. krok převod na regulární výraz

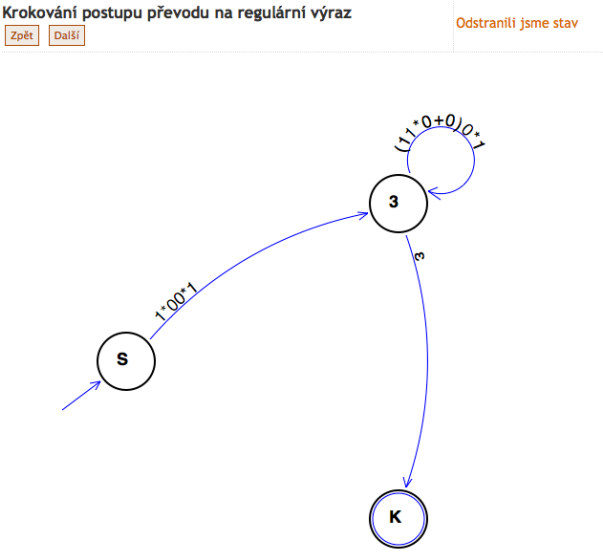


Obrázek 44: 4. krok převod na regulární výraz

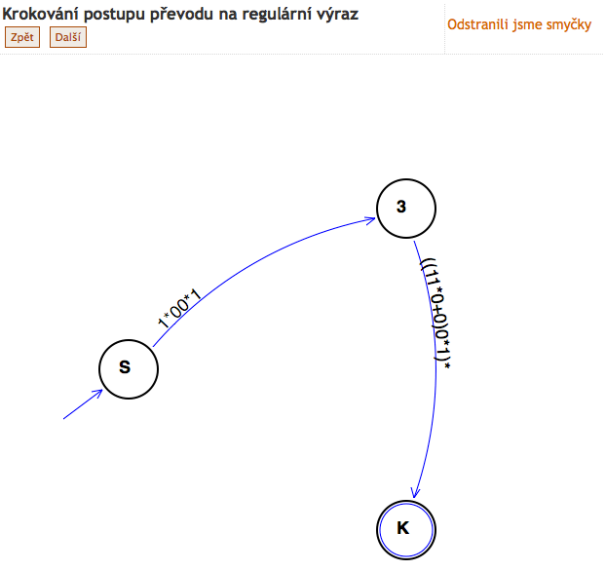
Na obrázku 44 došlo k odstranění stavu. Byl odstraněn stav 1. Na tomto místě musím osvětlit, proč se odstranil právě stav 1, jelikož stavy se neodstraňují popořadě, jak by se mohlo zdát. Při odstraňování stavů vybírám vždy ten stav, u něhož je součet hran, které do stavu vstupují a hran, které ze stavu vycházení, co nejmenší. Důvod je poměrně jednoduchý. Při odstranění stavu se vezme řetězec znaků, kterým je ohodnocena hrana vstupující do stavu. Řetězec se kopíruje tolikrát, kolik hran ze stavu vychází. Čím méně tedy hran je, tím méně řetězců je nutné kopírovat.



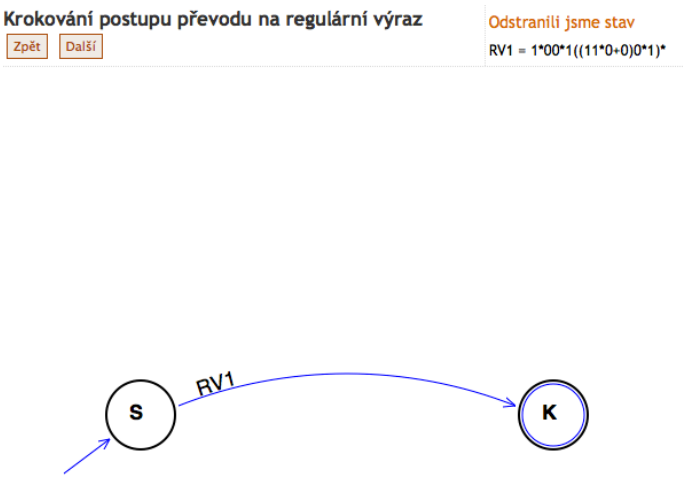
Obrázek 45: 5. krok převod na regulární výraz



Obrázek 46: 6. krok převod na regulární výraz

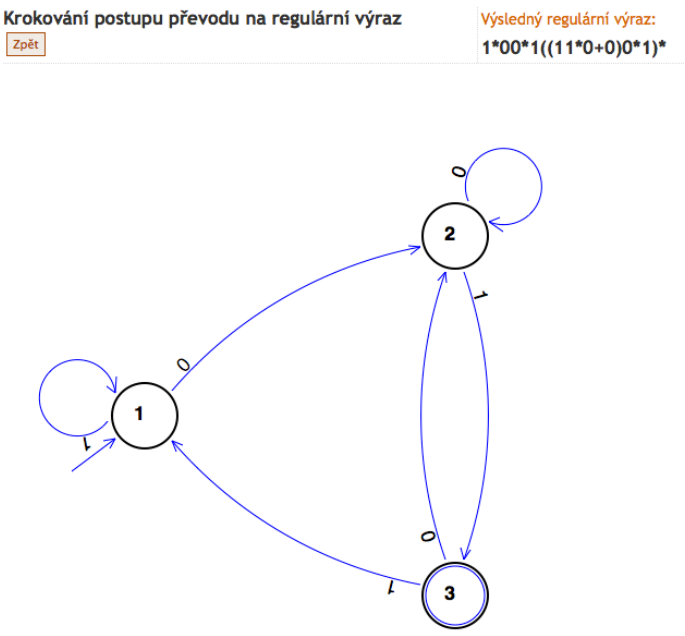


Obrázek 47: 7. krok převod na regulární výraz



Obrázek 48: 8. krok převod na regulární výraz

Na obrázku 63 je vidět, že hrana se místo konkrétním regulárním výrazem ohodnotila výrazem RV1. Tento výraz je v horní části vysvětlen. Tuto úpravu jsem musela udělat, protože na hraně vznikl odstraněním stavu 3 velmi dlouhý regulární výraz, který by se na hranu nevešel a tak jsem v zájmu zachování názornosti musela tento výraz nahradit a vypsat ve vysvětlivkách v horní části.



Obrázek 49: 9. krok převod na regulární výraz

Obrázek 64 znázorňuje poslední krok algoritmu převodu na regulární výraz. V horní části obrazovky je vypsán hotový regulární výraz, který vznikl ze zadaného automatu. V hlavní části je zobrazen graf původního automatu, aby měl uživatel možnost porovnat si výsledný regulární výraz s automatem.

2.10 Krokování a animace algoritmů

2.10.1 Krokování

Krokování algoritmů jsem naprogramovala pomocí PHP. Krokování probíhá pomocí tlačítek zpět a další. Samozřejmě tlačítko zpět není aktivní u prvního kroku a tlačítko další není aktivní na posledním kroku.

Po přechodu na stránku algoritmu se všechny kroky tohoto algoritmu postupně provedou a uloží do pole. Dále si uchovávám údaj o tom, ve kterém kroku algoritmu se právě nacházím. Při posunu dál zvýším krok o jedna a zobrazím příslušnou položku pole. Při posunu zpět krok o jedna snížím a opět zobrazím příslušnou položku pole. Toto řešení umožňuje uživateli projít si celý algoritmus tam i zpět, což je výhodné pro výukovou aplikaci.

2.10.2 Animace

První variantou, kterou jsem zkoušela, když jsem řešila animaci algoritmů, bylo programovat animace pomocí SVG. Bohužel jsem narazila na problém jeho podpory v různých prohlížečích. Od výukové aplikace se samozřejmě očekává, že bude fungovat pro všechny prohlížeče, protože studenti využívají různé operační systémy a s nimi i různé prohlížeče.

Další variantou bylo použití JavaScriptu, ale vzhledem k tomu, že celá aplikace je napsána v jazyce PHP, jsem i tuto možnost zavrhla, abych zachovala jednotnost aplikace.

Nakonec jsem se rozhodla pro animaci pomocí HTTP hlavičky. Při animaci vytvořím pole se všemi kroky algoritmu, jako bych dělala krokování. Místo, abych se mezi jednotlivými kroky algoritmu posouvala pomocí tlačítek, aktualizuji stránku pomocí HTTP hlavičky, přičemž si uchovávám aktuální pozici a při každém obnovení se posouvám na další krok. To opakuji, dokud neprojdu všechny kroky algoritmu. Mezi jednotlivými kroky jsem nastavila prodlevu 3 sekundy, což je podle mne dostatečné, aby se uživatel stihl zorientovat, a přitom dlouho nečekal na další krok.

2.11 Nápověda

Aby mohla být aplikace využívaná jako výuková, musela jsem ji doplnit nápovědou. Zvažovala jsem několik možností, jak nápovědu udělat.

První možností byla nápověda umístěná do souboru PDF, kterou by si každý uživatel mohl stáhnout na hlavní stránce aplikace, a měl by v ruce manuál, který by mohl kdykoli použít. Možnost jsem nevyužila, protože je lepší nezatěžovat uživatele stahováním a listováním v souboru, když mu mohu nabídnout nápovědu k tomu, co právě vidí před sebou. Další nevýhodou tohoto řešení je fakt, že aplikace nefunguje v režimu off-line, a proto stažená nápověda bude uživateli jen na obtíž.

Druhou možností bylo vytvořit nápovědu, která se zobrazí v novém okně pomocí odkazu na každé stránce. Právě kvůli nutnosti přejít na novou stránku jsem radši zvolila jiné řešení.

Volbou, pro kterou jsem se rozhodla, bylo vytvoření nápovědy jako popisu, který se zobrazí, když umístíme kurzor myši na obrázek otazníku. Tak má uživatel možnost podívat se na vysvětlení kdykoli, bez nutnosti přecházet mezi stránkami.

Závěr

Vytvořila jsem webovou aplikaci, díky níž lze pracovat s deterministickými i nedeterministickými konečnými automaty. Uživatelé je mohou zadat prostřednictvím formuláře nebo souboru. K dispozici jsou i ukázkové příklady. Konečný automat přehledně vykresluji. Vytvořila jsem grafické ztvárnění algoritmů minimalizace a převodu konečného automatu na regulární výraz.

Nad rámec zadání jsem přidala zobrazení algoritmu pro převod nedeterministického konečného automatu na deterministický, protože bez tohoto algoritmu by podle mne byla aplikace neúplná.

Uživateli dávám možnost nejen krokovat všechny algoritmy, ale také spustit je jako animaci.

Hlavním přínosem práce je právě přehledné zobrazení algoritmů, které se probírají ve výuce předmětu Automaty a formální jazyky. Věřím, že pomůže studentům, kteří budou aplikaci využívat, k rychlejšímu pochopení problematiky, a tak jim usnadní učení.

Možnost rozšíření aplikace spočívá rozhodně v přidání dalších algoritmů z teorie formálních jazyků.

Seznam obrázků a tabulek

Použitá literatura

[1] Timothy Boronczyk, Elizabeth Naramore, Jason Gerner, Yann Le Scouarnec, Jeremy Stolz, Michael K. Glass: *PHP 6, MySQL, Apache Vytváříme webové aplikace*, Computer Press, Brno, 2011, ISBN 978-80-251-2767-4

[2] Prof. RNDr Milan Češka, CSc.: *Teoretická informatika Učební texty*, FIT VUT, Brno, 2002

[3] RNDr. Miroslava Dubcová, Ph.D. RNDr. Lucie Purmová, Ph.D., doc. RNDr. Carmen Simerská, CSc.: *Sbírka příkladů z matematiky II ve strukturovaném studiu*, vydavatelství VŠCHT, Praha, 2008, 1. vydání, ISBN 978-80-7080-706-4, strana 130.

[4] Eichlerová Kateřina: *Vizualizace konečných automatů pro výukové účely*, Diplomová práce, FP TUL, 2011.

[5] Eisenberg J. David: *SVG Essentials*, O?Reilly 2002, ISBN: 0-596-00223-8

[6] PaedDr. Hashim Habiballa, PhD.: *Regulární a bezkontextové jazyky I,* Ediční středisko CIT OU, Ostrava, 2005, 2. vydání, ISBN 80-7042-852-X

[7] Hopcroft, Motwani, Ullman: *Automata theory, Languages and Computation* 3rd eddition. Addison Wesley, 2005. ISBN 0321455363.

[8] RNDr. Michal Chytil, CSc.: *Automaty a Gramatiky*, SNTL – Nakladatelství technické literatury, Praha, 1984, 1. vydání

[9] Prof. RNDr Petr Jančar, CSc.: *Teoretická informatika – učební text*, Ediční středisko VŠB-TUO, Ostrava, 2007, 1. vydání, ISBN 978-80-248-1487-2

[10] Petr Jančar, Martin Kot, Zdeněk Sawa: *Teoretická informatika – učební text*, Ediční středisko VŠB-TUO, Ostrava, 2007, 1. vydání

[11] Kosek Jiří: *PHP a XML*, Grada Publishing 2009, ISBN 978-80-247-1116-4

[12] Kosek Jiří: *PHP – tvorba interaktivních internetových aplikací*, Grada Publishing, Praha 1999, 1. vydání, ISBN 80-7169-373-1

[13] Vaníček Jiří, Papík Martin, Pergl Robert, Vaníček Tomáš: *Teoretické základy informatiky*, Kernberg Publishig, 2007. ISBN: 978-80-903962-4-1.

[14] Vrána Jakub: *1001 Tipů a triků pro PHP*, Computer Press Brno 2010, ISBN 978-80-251-2940-1

[15] Jiří Kosek, *XML schémata* [online], 2005-8 [citace 2012-05-08], dostupné na: http://www.kosek.cz/xml/schema/index.html

[16] *PHP.net* [online], 2012-05 [citace 2012-05-08], dostupné na: http://www.php.net/

[17] *eXtensible Markup Language (XML)* [online], 2012-01 [citace 2012-05-08]. Dostupné na: http://www.w3.org.XML/

[18] *State Chart XML (SCXML)* [online], 2012-02 [citace 2012-05-08]. Dostupné na: http://www.w3.org/TR/scxml/

[19] *Relax NG* [online], 2011-01 [citace 2012-05-08]. Dostupné na: http://relaxng.org/

[20] Eric W. Weisstein, *Circle* [online], 2012-04 [citace 2012-05-08]. Dostupné na: http://mathworld.wolfram.com/Circle.html/

Příloha A: Obsah přiloženého CD

Adresář (soubor)	Obsah
/ZdrojoveKody	Zdrojové kódy aplikace
DPCernikova.pdf	Text diplomové práce ve formátu PDF
DPCernikova.docx	Text diplomové práce ve formátu DOCX